# TECHNICAL UNIVERSITY - SOFIA

## ENGLISH LANGUAGE FACULTY OF ENGINEERING

# RESEARCH PROJECT

## MEng DEGREE

**TITLE: Modelling and application of μ-synthesis for hovering quad-rotor**

**SUPERVISOR: Prof. P. Petkov** **STUDENT: Unai Garcia Artola**

**SOFIA 2013**

# ***Index***

# *Abstract*

It is known that nowadays Control Engineering is more and more used due to the utility in the development that is has in different branches of the engineering. That utility is based in the simplicity in the representation of the different elements and systems that make easier the development of the work.

As it will be explained in the following chapter, inside of the control engineering there is more than one branch. In the following paper the application of the Robust Control to a small quad-rotor is explained, and specifically the implantation of μ-synthesis on a quad-rotor, in order to get a controller which it will help to the device to fly in a straight and level flight.

For that aim, firstly a short review of the history of the Control Engineering is made. After, deep explanation of Robust Control and commands that are going to be used in the development of the work are shown, in order to introduce the elements that are used in the following work.

In the second chapter the device that is used for the application of the μ-synthesis is explained. As it was explained in the paragraphs above, the device that is chosen is a small quad-rotor. In the chapter dynamics and cinematic models are explained. After, Simulink and linearization of the system are shown.

It is in the third chapter where μ-synthesis is explained deeply in order to see how it works in the one of the controller that is applied. After the explanation, different controllers that are got will be shown, in order to see which one of the gets in better way the aim of this work. At the same time, with the data that are got in the development, it will be possible see if the system got is close to the reality and it can be used to simulate or not.

After getting the appropriate controller and check that the simplification of the system is correct, in the fourth chapter wind disturbances and noises will be introduced in the Simulink model in order to make the simulation more realistic.

Finally in the last chapter, after the discussion of the result, conclusions about the work are shown in order to see if the main of the project is got or not.

Also, an annex can be found in the last pages with all the Matlab functions that are used in the development of the work

# Chapter 1: Control Engineering and Robust Control

## 1.1 Introduction

On the last recent years the interest of the development of the Robust Control System on UAV (Unnamed Aerial Vehicles) has increased due to the good results that were got with the application of this kind of control.

Nowadays there are two main techniques to get robust controllers, $H_\infty$ design and μ-synthesis. The first one is preferable to use due to the good results that are got in controllers order, which normally are from lower order than the controllers that are got in μ-synthesis. Inside the $H_\infty$ there is more than one method and the most used method is the one called Loop-Shaping, but also Mixed-Sensitivity design is used.

But the fact of using the $H_\infty$ design does not ensure robust performance in the general case of structured and unstructured uncertain. For avoiding this problem, is better to use the μ-synthesis, which is based in the use of the structured singular value may ensure robust stability and robust performance in the presence of exogenous disturbances, noises and different type of uncertainties. For that, is necessary a linearized model of the quad-rotor dynamics and also may include some weighting functions that shape the exogenous signals and represent the performance requirements to the closed-loop system

On the following pages the application of the μ-synthesis on small quad-rotor on a hovering position will be explained. But first of all, it is necessary to introduce a bit History about Robust Control and also some theory of it. Once that Robust Control is introduced, the following step is the study of the commands that will be used in Matlab for the development of the μ-synthesis for the small quad-rotor.

### *1.1 Control Engineering*

The Control is an area from the engineering which is included in the Engineering Control. But, what is Control Engineering? Is an interdisciplinary approach for controlling devices and systems, which is used in several areas like: electric, electronic, mechanic, chemistry, process engineering… But nowadays, electric and electronics are the nearest areas due to the easy way of building model using techniques from Control Theory. Then, all these models are used in flight systems and lately in automotive industry.

Control has different subdisciplines depending on the type control. These are some of them:

- Open-loop control
- Closed-loop control
- Regulation – Set-point control
- Linear control
- Nonlinear control
- Optimal control
- Robust control

### *1.1.2 History of Control*

There can be appreciated four different ages in the development of the history of control:

1- The development made by Greek and Arabs. From about 300 BC to about 1200 AD.

2. The Industrial Revolution in Europe. In the third quarter of the eighteenth century.

3. The beginning of mass communication and the First and Second World Wars. This represents period from about 1910 to 1945.

4. The beginning of the space/computer age in 1957.

It's believed that the first application of Control appeared en ancient Greek, where Ctesibius around the 240 B.C. built a water clock with a regulatory system with float like the same of the picture 1. This clock works in the following form:

Through the tube M water enters to the clock - assuming a constant water flow and equidistant hours. Water falls to the main tank, where the figure C is floating and pointing the hours with a spear on a cylinder. As the water falls into the tank and the level rises, the figure goes up and dialling hour after hour in the cylinder. But it is not all, when the main deposit is full, the water falls through the tube (siphon) FBE to the other tank and the figure C goes down for starting again counting hours. When the deposit which is down is full, it turns and in the same time the gear turn making roll a system of gears, which rotates the cylinder. The marks of this cylinder are not at same distance; they are done with come curvature for setting to the day duration. This cylinder completes one turn in 365 days.

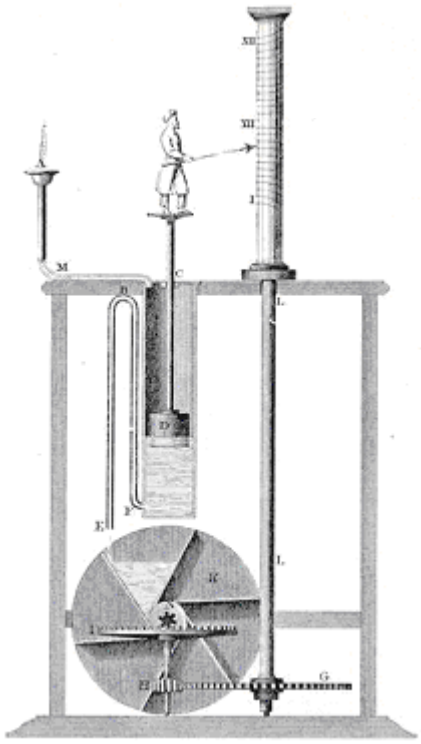


Figure 1: Ctesibius clock

Figure 2: Heron

Many years after this clock, Heron of Alexandria (10 A.D.-70A.D.) published a book called "*Pneumatica"* where there are described several mechanism of float regulators. He also published which is considered the first book of robotics called *"Automats"*.
In the figure 3 is shown the magic fountain developed by Heron which works automatically. This is considered another application of control.

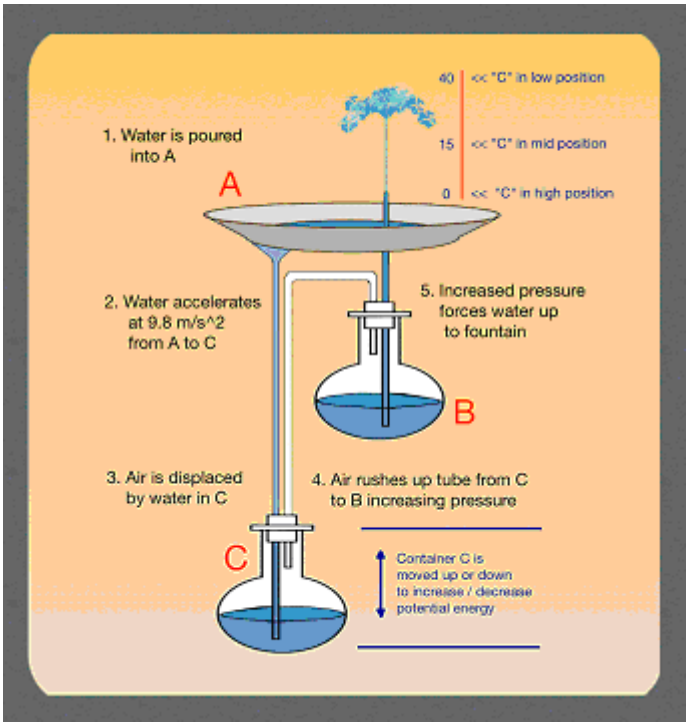



Figure 3: Magic fountain of Heron

It was in XVI Century when it was developed the first control system with feedback. It was made by a Dutch called Cornelis Drebbel (1572-1634). He built an incubator with an explicit feedback for regulating the temperature. But the most important achievement of Drebbel was the first useful submarine with included feedbacks systems also.

Some years after, a French physicist and mathematician called Denis Papin (1647-1712) invented the first automatic pressure regulator for steam boilers (figure 4) which nowadays it's applied in pressure cookers.

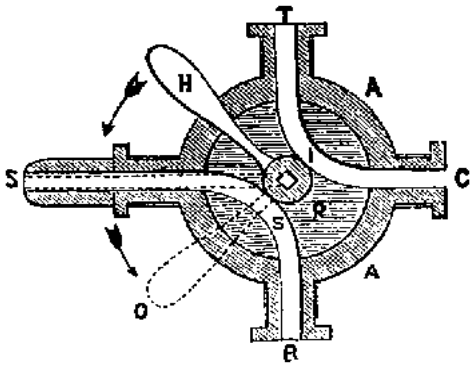



Figure 4: Pressure regulator

However, the first important work in control with automatic feedback appeared in 1788 with the invention of the centrifugal flyball governor used for regulating the speed of steam engines (figure 5) developed by James Watt (1736-1819).
It was made with two or more masses which rotate around a rotating shaft. As a result of the centrifugal forces, the masses tend to move away from the rotation spindle. If the motion goes far enough, this motion causes the lever arms to pull down on a thrust bearing, which moves a beam linkage, which reduces the aperture of a throttle valve. This valve, which is connected to the governor who receives the power from the engine's output shaft, regulates the flow of the working fluid which supplies the prime mover. This working fluid enters in the cylinder changing the speed of the prime mover.

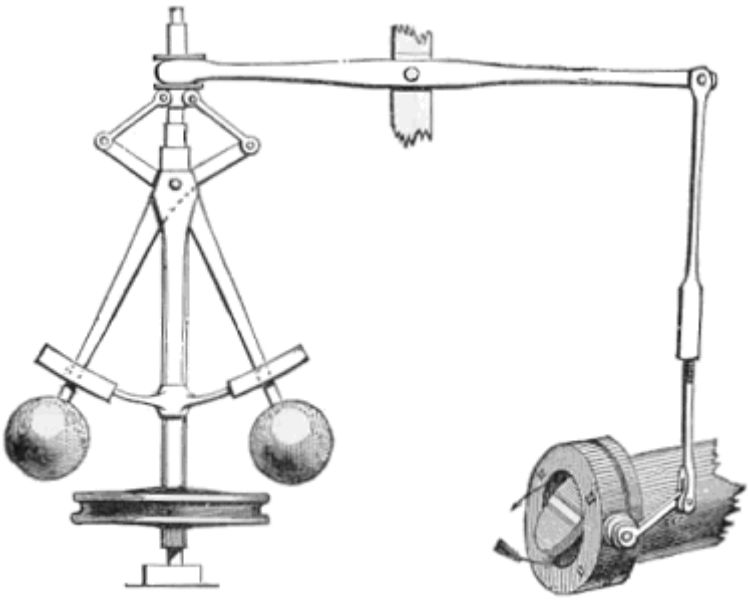


Figure 5: Centrifugal flyball governor

Until the late nineteenth century the automatic control was characterized as an intuitive science. But the desire of developing the transient responses and correctness of the Control Systems, forced to develop the Control Theory.

It was many years later, when a Scottish physicist called J.C. Maxwell (1831-1879) relate the mathematical theory and the control theory using the model of differential equations. It was in 1868 when he contributed with several points like:

- Stability concept
- Simple mathematic models
- The importance of the integral action
- Linearization
- Stability as an algebraic problem
- Stability Criterions for systems of first, second and third order.

Roughly, Maxwell developed a technique which linearizes the differential equations of motion to find the characteristic equation of the system. This equation has some parameters which if they are changed, the behaviour of the system changes, and therefore, the stability of the system changes. Maxwell came to the conclusion that if the roots of the characteristic have real part negative, the system is stable. With the development of this achievement it is considerer that the theory of control systems was firmly established.
In 1877, a Russian scientist called I. Vyshnegradsky (1830 - 1895), developed the analysis of the stability of regulators using differential equations, but he did it independently from Maxwell.

In the same year, a numerical technique for determining when a characteristic equation has stable roots was built by the English mathematician E. Routh (1831-1907), which nowadays is widely used and studied in universities by the students of engineering.

The work done by the Russian astronomer A. Lyapunov (1857-1918) when he did it, it did not seem important, but many years after, it become very important, because the development of the studies of the stability in non-linear systems was very useful for the development of other theories.

The British engineer O. Heaviside invented operational calculus in 1892-1898. He studied the transient behaviour of systems, introducing a notion equivalent to that of the transfer function.

The beginning of the twenty centuries carried with him another big change in Control Engineering due to two reasons:

- The development of the telephone and mass communication.
- World Wars.

The mayor problem of the mass communication in long distances is that it needed to be amplified periodically, but with this amplification, the noise is also amplified.
In 1927, the American electrical engineer H.S. Black (1898-1983) introduced the negative feedback amplifier (figure 6) for reducing the distortion. In the field of electronics this feedback is very useful once the stability problem of the system is solved.

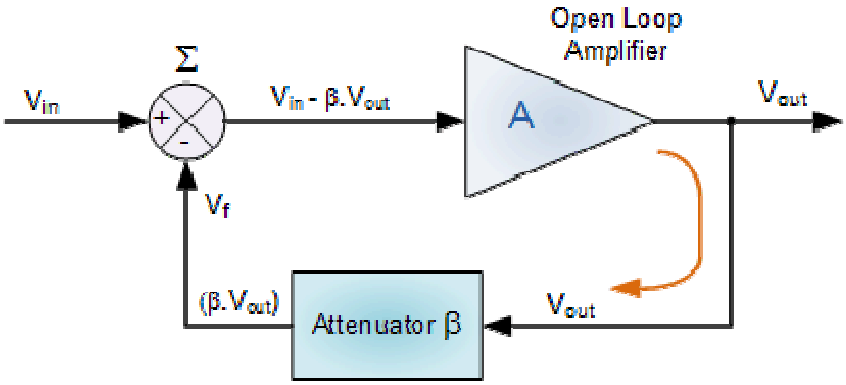



Figure 6: Feedback amplifier

In 1932 the Swedish electronic engineer H. Nyquist (1889-1976) creates a simple method for determination the stability on closed-loop system called *Nyquist Stability Criterion* which nowadays is used.

Some years after Nyquist achievement, H.W. Bode (1905-1982) created an innovative method, which studying time domain stability using the frequency domain concepts of gain and phase margin, the stability of the system could be studied. For that he developed his own plots called asymptotic phase and magnitude plots, which they received his name.

In the end of the decade of the 40s early 50s, the American electrical engineer W.R. Evans (1920-1999) invented a graphical method to study how the roots of a systems change varying certain parameters of the system. This method called Root Locus Analysis is used to determinate the stability of the system.

World Wars carried with them an intensive development of Control Systems. In the one hand, there were some problems with ships navigation control which it was solved by the Russian mechanical engineer N. Minorsky (1885-1970), who propose to use PID (Proportional-Integral-Derivative) controller in the automatic steering systems for U.S. Navy ships. This controller considers the nonlinear effects of the closed-loop system.
In the other hand, there was a big problem with the accurate pointing of weapons on board of ships and aircrafts; for solving that The Norden bombsight, developed during World War II, used synchro repeaters to relay information on aircraft altitude and velocity and wind disturbances to the bombsight, ensuring accurate weapons delivery.

The fourth most important age came with the development of the computers which helped to create modern control. Due to the availability of digital computers, became possible the analysis of complex systems in the time domain. Since then, the Modern Control Theory started developing more and more. Modern control theory is based on the analysis and synthesis in the time domain. Using state variables. These variables describe the mathematical state of a dynamic system (system which changes as time is passing).

## *1.2 Robust Control*

If in all the branches of the control theory is one which works with uncertainties in the parameters of the system, Robust Control is one of the most important of them. The previous methods of Robust Control like state-space, were not sometimes good due to the lack robustness. This is the reason why the Robust Control was developed, for getting robust stability and robust performance in the presence of bounded modelling errors. In contrast with an adaptive control policy, a robust control policy is static; rather than adapting to measurements of variations, the controller is designed to work assuming that certain variables will be unknown but, for example, bounded.

It was in 1927 when appears for first time the idea of Robust Control, due to a patent developed by H.S. Black in which the Robust Control problem appeared. Black proposed feedback and large-loop gains for the design of an accurate system given significant plant uncertainties. But the problem was that most of the systems designed in this way they were unstable. With the results got by Nyquist in 1932 it was when they came to the conclusion of the relation between the dynamic stability and large-loop gain. Both things, Nyquist frecuency-domain stability criterion and Black's concept of large loop gain for system accuracy, established the basis of Robust Control design.

Between 1960 and 1975 there was another period in which the Robust Control was developed in an important way. This period was called *State-Variable Period* due to the introduction of a new state-variable concepts like controllability, observability, optima linear quadratic state feedback etc. Which were introduced by R.E. Kalman, electrical engineer born in Hungary in 1930. But the biggest results of this period were written in a book done by Anderson and Moore called *Linear Optimal Control* and published in 1971. Unfortunately, during this period, the problem of plant uncertainty was ignored.
It was in this period when they tried to introduce the concept of sensitive comparison matrix in MIMO (multiple inputs-multiple outputs). Some year ago this concept was n used in SISO (single input-single output) systems.

From 1975 to present it is considered the third period of the robust control. It was between the decade of seventies and eighties when the problem of the system uncertainties appeared again. At about the same time, some significant results were being reported on the analysis of multivariable systems in the frequency domain. For example, the stability criterion of Nyquist was generalized for multivariable systems by an English electrical engineer called H.H. Rosenbrock (1920-2010). This confluence of interest in uncertainty and multivariable systems led to the current period, which is referred like Modern Robust Control period (1975-present).

### *1.2.1 Reason of Robust Control*

The first step that it has to be done when a Control Engineering design is going to be done, is to create the mathematical model of a physic plant. In most of the cases, this model will be nonlinear and it will have high order. A model with a very high order is not so useful in the point of view of the dynamics, because the process of the design is complicated and then, results of the controllers that are got will have very high level. Due to those reasons, is tried to get the simplest model but it has to be similar to the original model; with same intrinsic characteristics and as complete as possible. It is clear that this model will has errors because is not the same as the original model, and because of that arises the question of if the controller that you get with the simple model is correct or not. This is reason because it was developed the theory of Robust Control.

In Robust Control it is tried to get an approximate linear model with constant coefficients, assuming that some errors are going to be. This error is considered as an uncertainty, which is modelled and bounded for getting the controller.

### *1.2.2 Introduction to Robust Control: basic notions*

To understand the robust control is necessary to explain which are different elements of the system and how the system works. In the different devices and systems the following concepts can be found:

1- Certain dependent variables called *outputs*, which need to be controlled for get behave that is wanted for the correct operation of the system.
2- Certain independent variables called *inputs*, which usually are measurements of variables of the system or only signals that are introduced to try to get the desired behaviour.
3- There are some unknown or unpredictable disturbances which could change behave of the system.
4- Uncertainties of the parameters of the system due to the simplify process that is made with the system. These uncertainties happened because the parameters are not known exactly.

The general plant is shown in the figure below this paragraph.

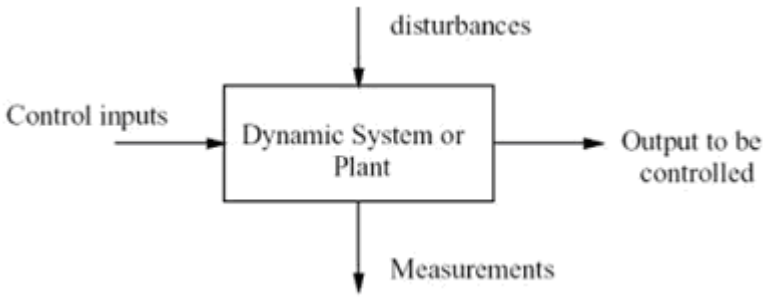



Figure 7: General plant of the system

But the control need a concept called *Feedback* who helps to the system works correctly. Feedback means that the outputs of the system are introduced to see how it is working and these outputs are compared with the reference inputs for getting the precise action which is necessary to apply to the system for achieving behaviour that is needed for the correct operation. In the figure 8 is shown the system (represented with *P*) with the controller (*K(s)*) and the feedback. The output is represented with *y*, the reference is *r* (is an input, like *y* that comes from the feedback) and the disturbance is *d* (which is an input too).

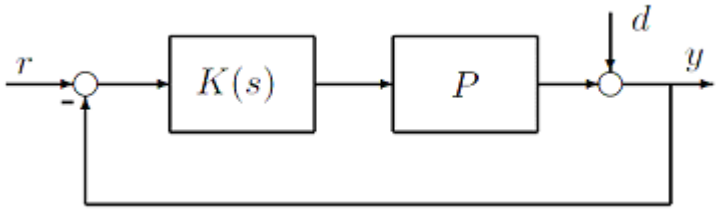



Figure 8: System with feedback

In theory of robust control the system is considered like an aggregate or families of models which are included in a nominal model *G(s)* with and bounded uncertainty. This uncertainty could be added to the system in different forms (additive uncertainty or multiplicative uncertainty). Depending of the way in which is done, the problem will have different treatment.

Once that the model is got, next step is the analysis of different concepts starting with the internal stability and finishing with the robust performance, which is the main aim of the robust control.

Internal stability

It is said that a system is internally stable if all the transfer functions get in all the pairs input-output have the poles in the negative part of the complex axis.
The feedback loop is stable if and only if the numerator of the transfer function of the feedback (1+G(s)*K(s)) is stable and there is not cancellation between the poles and zeros of the system and the controller in the right part of the complex semiplane.

### *1.2.3 LTI models*

Due to the uncertainties that can be found in systems because of the simplification of them, is necessary to know how to build uncertain models. This is one of the important steps in the design of control system. First is necessary to know how to built models of open-loop and closed-loop system in linear-time-invariant systems (LTI). Then, some functions for building the system with structured uncertainty (real) are showed, which can be found in Matlab toolbox called Robust Control ToolBox®3. Finally, this part contains the explanation of how to build system with unstructured uncertainty (complex). During this part, two different uncertainties will be considered, additive and multiplicative.

The creation of the linear-time-invariant system models is done with the following commands in Matlab:
- *frd* → Frecuency response data models.
- *zpk* → Zero-pole-gain models
- *tf* → Transfer function matrices
- *ss* →State space models

The model can be built in different forms. If the model is described with an equation system the system is built in the way that is described in *A form*, and if the system is described with the transfer function matrix, the system is built in the form that is described in *B form.*

***A form***

The state space model is created with the following commands.
*Gss = ss(A,B,C,D)*

For displaying one of the matrixes, A for example:
*A = Gss.A*

The state space model can be converted to transfer function with the command:
*Gtf = tf(Gss)*

For knowing the poles and transmission zeros, is necessary to write the commands below:
*p = poles(Gss)* for poles and *z = zeros(Gss)* for zeros

In the similar form, for knowing the poles and zeros of the transfer function matrix model:
*p = poles(Gtf)* for poles and *z = zeros(Gtf)* for zeros

With the command *G = ss(Gtf)* is produced the state space realization from the transfer function matrix. But sometimes, the system that is got is not the minimal. For getting the minimal system is necessary to modify a bit the command above. With the following command is get the minimal state space realization
*G = ss(Gtf,'min')*

In the case that is wanted to create a discrete state space system, the command change has to be changed in the form below:
*Gd = ss (A,B,C,D, Ts)*

And in the case of discretization of a continuous state space model:
*Gd = c2d(G,Ts)*

Where Ts is the sampling time in both cases.

Once that system is described, if it is wanted to plot the singular value of the system frequency response between to values of time, it will be useful the command sigma. The form of using the command is the following:

*Sigma(G,{10^(-2) 10^3})*

With this command is get the plot of G between 10^-2 and 10^3.

If the system is composed of two transfer matrices G1 and G2, in the case that they are connected in parallel , the general system will be considered G=G1+G2, and in case that the two system are connected in row, the general system is considered G=G1*G2:

***B form***

In the case that the system is given with the transfer function matrix, the state space system is built in the following form (two input-two output example):

*s = tf('s')*
*g11* = (transfer function corresponding with the (1, 1) element of the matrix)
*g12* = (transfer function corresponding with the (1, 2) element of the matrix)
*g21* = (transfer function corresponding with the (2, 1) element of the matrix)
*g22* = (transfer function corresponding with the (2, 2) element of the matrix)
*G = [g11 g12;g21 g22]*

The rest of the command explained in the A form, can be use also in B form.

If it is wanted to know the largest singular value, the command that is used is:
*norm(G,'inf')*

Now, if it is considered a multivariable feedback system as in figure 9, where are shown a reference *r*, input disturbance *di*, output disturbance *d* and sensor noise *n*. The plant of the system is represented with the letter G and the controller with K.

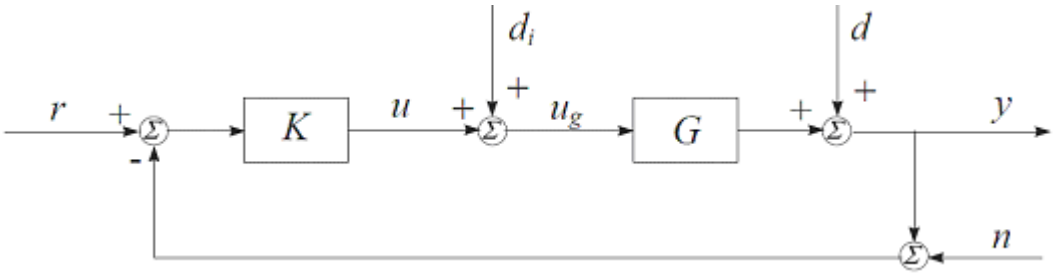



Figure 9: Multivariable feedback

In case of the system is internally stable, these following equations will be satisfied:

$$y = To*(r - n) + G*Si*Di + So*d \quad (1.1)$$
$$r - y = So*(r - d) + To*n - G*Si*Di \quad (1.2)$$
$$u = K*So*(r - n) - K*So*d - Ti*di \quad (1.3)$$
$$u_g = K*So*(r - n) - K*So*d + Si*di \quad (1.4)$$

For understanding these equations, is necessary to explain some concepts.

*Li* and *Lo* are input loop transfer and output loop transfer respectively. They are got in the way below:

$$Li = K*G \quad (1.5)$$
$$Lo = G*K \quad (1.6)$$

*Si* represents the input sensitivity matrix, defining transfer function matrix from *di* to $u_g$:

$$Si = (I + Li)^{-1} \quad (1.7)$$

*So* represents the output sensitivity matrix, defining transfer function matrix from *d* to y:

$$So = (I + Lo)^{-1} \quad (1.8)$$

Finally, the output and input complementary sensitivity matrices are defined in the following form:

$$Ti = I - Si = \frac{Li}{I - Li} \quad (1.9)$$

$$To = I - So = \frac{Lo}{I - Lo} \quad (1.10)$$

The sensitivity transfer function matrices and corresponding frequencies are called also *sensitivity functions.*

The commands for all these elements are the following:

```
Looptransfer = looptranfer(G,K);
Si = looptransfer.Si;
So = looptransfer.So;
Ti = looptransfer.Ti;
To = looptransfer.To;
Li = looptransfer.Li;
Lo = looptransfer.Lo;
```

It is possible to get the poles of the closed loop transfer function with the command:
*p = looptransfer.Poles.*

The command *stab = looptransfer.Stable* will return 1 if the system is stable and if is not, the answer will be a 0.

### *1.2.4 Structured uncertain models*

- Uncertain real parameters

In case that is not known the exact value of the parameters, is necessary to define the parameter between some values, which help to the program to get the controller easier.
For introducing those parameters in the program this is the command that it has to be introduced:
*p = ureal('name',Nominal value,'Property 1',Value 1,… ,Property 2',Value 2,….)*

In this case, *p* is the name of the parameter. In *name*, we write the name of the parameter, then the nominal value. There are different properties when a parameter is going to be defined:
*Plusminus* → It means the absolute deviation from the nominal value.
*Range* → Is an interval containing the nominal value.
*Percentage* → The percentage deviation from the nominal value.

- Uncertain Space-State Systems

Once that the uncertain parameters are defined in the first lines of the program, now is possible to define the state-space system. This can be done in more than one way.

- A form

If matrices which define the system are known, the state-space system can be built in the following form:
*uss = ss(A,B,C,D)*

Sometime there are some parameters that are repeated more than one, so is necessary to simplify the system with this commands:
*simplify(uss,'full')*

- B form

There is another option, in the case that is known the transfer function; In this case space-state system can be built like this:
*ss = tf(num,den)*
Where *num* is the numerator of the transfer function and *den* is denominator of the transfer function.

- C form

Finally, there is a third form of defining the state-space system. With the block diagram, is possible to built using the function *feedback*.
The following example is from a mass-damper-spring system with the block diagram of the figure 10:

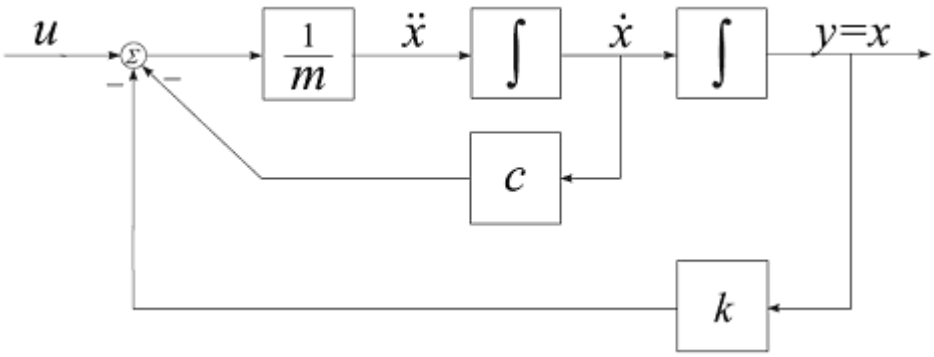



Figure 10: Mass-damper-spring system

*s =tf('s');*
*g1 = (1/s)/m;*
*int2 = 1/s;*
*uss = feedback(int2*feedback(g1,c),k);*

If it is wanted to see the properties of the state space system, they can be got typing the following command:
*get(uss)*

Sometimes is necessary to substitute the uncertain values for other specifies values. This can be got using the command *usubs*.
*B = usubs(name of the uss,'parameter1',value1,'Parameter 2',value2,....)*

There is another useful command for uncertain objects, this command is *usample*, which take randomly examples of the system.
*B = usample(A,n)*
This command take *n* random examples of the matrix A, and save them in an array called B with [size(A) , n] dimension.

For plotting bode diagrams of the uncertain system, these are the commands which have to be typed:
*w logspace(-1,1,200);*
*figure 1*
*bode(uss,w)*

Typing the command *step(uss)* is got the step response of the state-space system for 20 random values.

The command lines
*frres = frd(uss,w);*
*nyquist(frres)*
produce the nyquist diagram of the *uss* state-space for 21 different values, 20 are randomly chosen and the last one is the nominal value.

In robust control, sometimes there are cases in which the uncertain parameter can not be defined like real parameter. In these cases, the parameters are complex, so they have to be defined with different command. The command is similar to the command for defining the real parameters, only has two differences:

*p = ucomplex('name',Nominal value,'Property 1',Value 1,... ,Property 2',Value 2,....)*

In the case of complex parameters, the uncertainty is described in two different ways:
*Radius* → Radius of disk centred in the nominal value.
*Percentage* → Disc size in percentage of magnitude of nominal value.

The rest of the commands explained above can be used in the same way as for the normal parameters.

- Discomposing uncertain objects

It is possible to discompose the uncertain objects (*uss, ufrd, umat*) in two different parts as in the figure 11. One is linear fractional transformation of non-uncertain part and the other is a matrix containing the uncertain parameters. This is made by the command *lftdata*. Typing the following commands, the system is discomposed in two parts.
*[Gnom, Delta, Blkstruct, Normunc] = lftdata(usys)*

The system is defined with *usys*. *Gnom* is the non-uncertain part, *Delta* is the matrix which represents the uncertain part, *Blkstruct* returns n x 1 structure where *Blkstruct(i)* describes the i-th normalized uncertain element and *Normunc* returns an array of normalized uncertain elements.

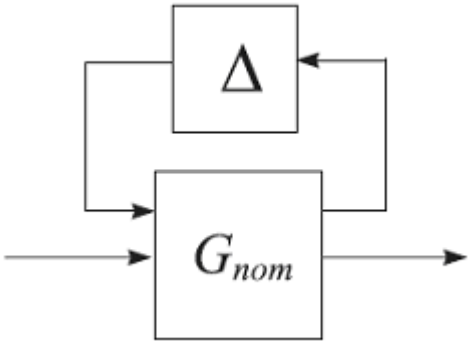



Figure 11: Decomposition of uncertain object

Another form of building uncertain models

***- Other forms of building uncertain models***

There are another two different forms of building uncertain models which are going to be explained here.

- A → *iconnect*

This is used to for building complex interconnections of uncertain matrices and system. The command *iconnect* has three different fields that are going to be defined by the user:

- Input → symbolic column vector, represents the input variables
- Output → symbolic column vector, represents the output variables;
Equation → cell-array of equality constraints (created using the command equate) describing the relations between input, output and intermediate variables.

The input, output and intermediate variables should be objects of class *icsignal*. They are symbolic vectors representing the signals in the interconnection.
For instance, the command
*icsignal(4)*
creates a four element column vector for use by *iconnect*.

For understanding easier how to use this command, in the next line will be explained an example of the mass-damper-spring system with the *iconnect* command.

In this case, *u* is the input and *x* is the output. *x* and *xdot* are used like intermediate variables and c, k and m are uncertain parameters which they have to be defined in the program before these lines.
Then, is necessary to define the equations which relation different variables with the parameters. In this case the equations are the following:

```
x = (1/s)* ẋ
xdot = (1/m*s)*(u – k*x-c*ẋ)
u = icsignal(1)
x = icsignal(1)
xdot = icsignal(1)
iconnect1 = iconnect;
iconnect1.Input = u;
iconnect1.Output = x;
iconnect1.Equation{1} = equate(x, tf(1,[1,0])*xdot);
iconnect1.Equation{2} = equate(xdot, tf(1,[m,0])*
(u-k*x-c*xdot));
```

- B → *sysic*

There is an alternative way of building complex interconnections of uncertain systems. With the command *sysic* it can be built also complex uncertain systems. Before using the command, is necessary to define 3 three variables: *systemnames, inputvar and outputvar*:

- *systemnames* is a *char* which contain the names of the system which have to be connected.
- *inputvar* is a *char* specifying the external inputs to the connection.
- *outputvar* is a char which contains the system outputs or the equations , which are used to produce the output variables.

Continuing using the mass-damper-spring example, the command lines in this case are the following:

```
m1 = inv(m);
int1 = 1/s;
int2 = tf(1,[1,0]);
systemnames = 'int1 int2 c k m1';
inputvar = '[u]';
outputvar = '[int2]';
input_to_int1 = '[m1]';
input_to_int2 = '[int1]';
input_to_c = '[int1]';
input_to_k = '[int2]';
input_to_m1 = '[u-c-k]';
uss = sysic
```

Is necessary to define the inverse of *m* with the command *inv(m)*, due to the uncertainty of m the direct division could create error.
To define interconnections must look at the block diagram.

### *1.2.5 Unstructured uncertainty model*

In case of unstructured (complex) uncertainty model, the function that it has to use is *ultidyn*. It means uncertain linear time invariant dynamics and it represents an unknown linear system whose is only known attribute is a magnitude bound on its frequency response. The command is used in the following form:

*H = ultidyn('Name',iosize,'Property1',value1,'Property2',value2,....)*

*Name* it has to be the name of the uncertain dynamic object. *Iosize* will describe the size with the number of outputs and number of inputs. The property Type specifies whether the known attributes about the frequency response are related to gain or phase and has value 'GainBounded' or 'PositiveReal', respectively. The default value is 'GainBounded'.

If Type is 'GainBounded', $\sigma\max[\Delta(\omega)] \leq \gamma$ for all frequencies, where $\sigma\max[\Delta(\omega)]$ is the maximum singular value of $\Delta(\omega)$.
When Type is 'GainBounded', the default value for Bound (i.e., $\gamma$) is 1.

If Type is 'PositiveReal', $\Delta(\omega) + \Delta^*(\omega) \geq \gamma$ for all frequencies, where $\Delta^*(\omega)$ denotes the Hermitian conjugate matrix of $\Delta(\omega)$.
When Type is PositiveReal, the default value for Bound (i.e., $\gamma$) is 0.

### *1.2.6 Types of uncertainties*

In this part two different types of uncertainties are going to be explained in a short form, these are usually used in robust control.

In case of additive uncertainty as in the figure 12, the model is chosen in the following form:

$$G(s) = G_n(s) + \Delta_a(s)*W_a(s)\ , \quad \text{where } |\Delta_a(s)| \leq 1 \qquad (1.11)$$

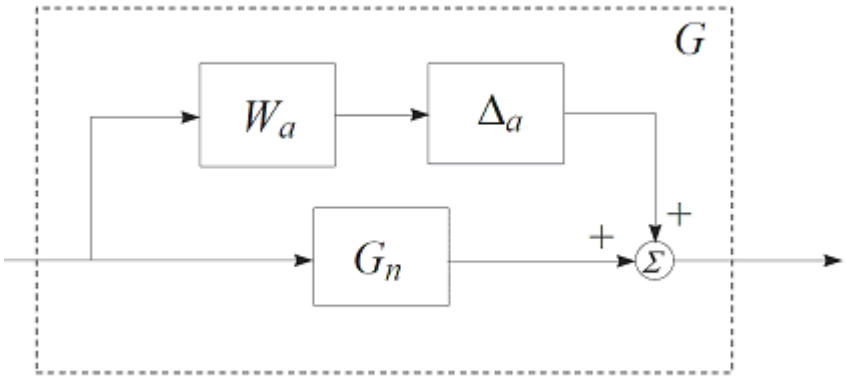



Figure 12: Plant with additive uncertainty

For additive uncertainty, the error is calculated as follows:

$$\left| G(jw) - G_n(jw) \right| = \left| \Delta_a(s) * W_a(s) \right| \leq W_a(s) \qquad (1.12)$$

If the uncertainty is multiplicative as in the figure 13, the model changes notoriously. In this case the model is chosen in the form below:

$$G(s) = G_n(s) * [\, 1 + \Delta_m(s) * W_m(s)] \,, \quad \text{where } \left| \Delta_m(s) \right| \leq 1 \qquad (1.13)$$

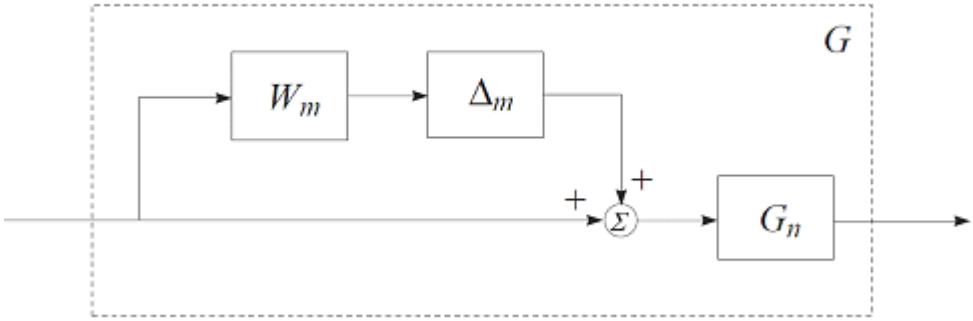



Figure 13: Plant with multiplicative uncertainty

For multiplicative uncertainty, error is calculated in a different way to that of the additive uncertainty. The way is the following one:

$$\frac{\left| G(jw) - Gn(jw) \right|}{\left| Gn(jw) \right|} \leq \left| Wm(jw) \right| \qquad (1.14)$$

### *1.2.7 Robust stability*

For understanding the objective of the robust stability, first is it useful to explain the uncertain control system as in the figure below (figure 14):

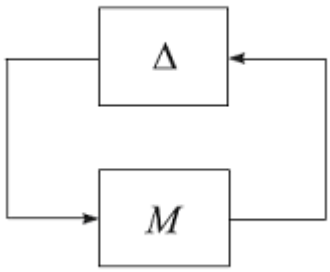



Figure 14: M - Δ loop for robust stability analysis

This loop transfer function is formed by two different parts. *M* is the nominal part which is separated from the uncertain part Δ. In the case of this last part, in the general case of structured uncertainty, it is used the singular value μ for studying the robust stability. In simpler case of unstructured Δ, is possible to use the small-gain theorem.

In most of the cases, is not possible to know exactly the value of μ, which depends of the frequency, but there are some algorithms which calculate approximately the value of μ with upper and lower bound, in structured singular values βu and βl respectively. This is the reason because the study of the robust stability has to be done by bounds.

• The uncertain system under consideration is guaranteed stable for all structured uncertain matrices Δ with $\| \Delta(s) \|_\infty < 1/\beta u$;
• There exists a specific structured transfer matrix Δ with
$\| \Delta(s) \|_\infty = 1/\beta l$ that destabilizes the system.

In addition, when the uncertainty is normalized

$$\| \Delta \|_\infty \leq 1 \qquad (1.15)$$

it follows that:

• If βu < 1, the system is robustly stable in respect to the modelled uncertainty;
• If βl > 1, the robust stability is not achieved;
• If βl < 1 and βu > 1, it is not possible to make sure conclusion about stability; it is possible that the system is not robustly stable.

The following command is that one that is used for the analysis of the robust stability. It can be used for structured uncertainty, unstructured uncertainty or mixed uncertainty.

*[stabmarg,destabunc,report,info] = robuststab(sys,opt)*

This command needs two input elements, which have been decided before tipping it.

*sys* → This part has to contain the system which is going to be studied his stability. It could be *uss* class or *ufrd*. In case of *uss*, the computations are done for appropriately chosen by command *robuststab* frequency values. In the other case, the associated with *sys* frequency vector is used.

*opt* → It contains optional input arguments.
Typing *opt = robopt('name1',value1,'name2',value2…)* an object is created for representing in *opt* command.

These are some of the properties which are mostly used:
*Display* → for displaying progress of the computations. By default is 'off';
*Sensitivity* → for computing the influence of individual uncertainties on the stability margin. By default is 'on'.

Output arguments:
*stabmarg* → is an structure containing the following elements:
*UpperBound* → upper bound of stability margin
*LowerBound* → lower bound of stability margin

*DesestabilizingFrequency* → frequency in which the system is destabilized. It corresponds with the upper bound of stability margin.

*destabunc* → structure containing a combination of uncertain parameter values closest to their nominal values that cause system instability. Corresponds to the upper bound of stability margin.

*report* → is variable which contains a text with results of robustness analysis.

*info* → structure with several fields:

*Frequency* → frequency vector used in computations.

*MussvBnds* → *frd* object containing the computed values of the upper bound (*MussvBnds(1,1)*) and lower bound (*MussvBnds(1,2)*) of the structured singular value.

*Sensitivity* → Structure with number of fields equal to the number of uncertain elements. In each field is stored a number, indicating the influence of the corresponding uncertain element on the stability margin. For instance, the number 40 means that if the uncertainty range is increased with 25%, the stability margin will decrease 10% (25% of 40).

### *1.2.8 Robust performance*

In the case of robust performance, the block diagram change a bit comparing with the case of the robust stability. The block diagram shown in the figure 15, which correspond with the block diagram of robust stability, contains and input element *v* and an output element *z*. The first one represents all exogenous signals to the system: references, disturbances, corresponding to the uncertainty structure of the system under consideration; the second one includes signals which have the meaning of "errors" which characterize the system performance. M contains the model of the nominal control system, weighting functions to represent the uncertainty and the weighting functions used to specify performance requirements. Δ represents the unknown part corresponding to the uncertainty structure of the system under consideration.

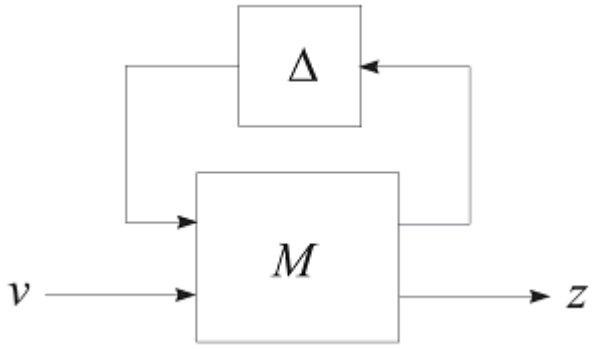



Figure 15: Block-diagram for robust performance analysis

Transfer function matrix from *z* to *v* is represented with $T_{zv}$ (s). It is appropriate as a system performance index to use the quantity

$$\left\| Tzv\,(s) \right\|_\infty \qquad (1.16)$$

Smaller values of this index means smaller "errors" z due to the "worst" input signals v and hence better system performance.

The problem of robust performance analysis may be reduced to the problem of robust stability analysis of the closed-loop which consists of the block M and the extended uncertainty block

$$\Delta p = \begin{bmatrix} \Delta & 0 \\ 0 & \Delta F \end{bmatrix} \tag{1.17}$$

In the equation above $\Delta F$ is a (unstructured) fictitious complex uncertain block, which his size is $n_v$ x $n_z$, where $n_v$ and $n_z$ are the sizes of *v* and *z* vector respectively. Using the structured singular value of M in respect of the extended uncertainty $\Delta p$, it will be got the robust performance. Due to the fact that is only possible to calculate the upper and lower bounds of the structured singular value, the conclusion for the robust performance should be made on basis to these bounds. The maximums in respect to the frequency of the upper and lower bounds of the structured singular value μΔP (M) are denoted by βu and βl, respectively.

• For all uncertainty matrices Δ with the given structure satisfying $\| \Delta \|_\infty < 1/\beta u$, the closed-loop system is stable and $\| Tzv(s) \|_\infty \leq \beta u$;
• There exists a specific matrix Δ with the given structure satisfying $\| \Delta \|_\infty = 1/\beta l$ for which either $\| Tzv(s) \|_\infty \geq \beta l$ or the system is unstable.

Usually, the weighting transfer functions that are used to specify performance requirements are chosen in a form which the following conditions is fulfil, the desired performance is get.

$$\| Tzv(s) \|_\infty < 1 \tag{1.18}$$

The system would get robust performance if the condition above is get.
For a performance requirement (10.3) and normalized uncertainty

$$\| \Delta \|_\infty \leq 1 \tag{1.19}$$

the following conclusions hold.

• If $\beta u < 1$ the system achieves robust performance for the modelled uncertainty (this includes also robust stability);
• If $\beta l > 1$ the robust performance is not achieved.
And if the case is that $\beta l < 1$ and $\beta u > 1$, is not possible to make a definite conclusion.

The case of the robust performance happens something similar to the case of robust stability, is necessary to introduce the concept of performance margin. The performance margin pm is equal to the reciprocal value of the maximum in respect to frequency of the structured singular value $\mu \Delta P (M)$, i.e., $pm = 1/\beta$. The upper bound *pmu* and lower bound *pml* of the stability margin are obtained by $\beta l$ and $\beta u$, respectively.

$$\text{pmu} = \frac{1}{\beta l} \qquad \text{pml} = \frac{1}{\beta u}$$

*robustperf* is the command used for the analysis of the robust performance. In a similar way of the robust stability, this command needs 2 input arguments and 4 different outputs arguments are got. The full command is the following:

*[perfmarg,perfmargunc,report,info] = robustperf(sys,opt)*

Input arguments:

*sys* → model of the uncertainty system. It could be continuous-time or discrete-time.
*opt* → optional input argument created by command *robopt* in the same way as for *robuststab*.

Output arguments:

*perfmarg* → this output is structured with the three following fields:
- *UpperBound:* upper bound of the performance margin.
- *LowerBound:* lower bound of the performance margin.
- *CriticalFrecuency:* frequency value corresponding to the upper bound of performance margin.

*perfmargunc* → a combination of uncertain element values corresponding to upper bound of performance margin.

*report* → char array with text description of the results from robust performance analysis.

*info* → structure with several fields:
*Frequency* → frequency vector used in computations.
*MussvBnds* → *frd* object containing the computed values of the upper bound (*MussvBnds(1,1)*) and lower bound (*MussvBnds(1,2)*) of the structured singular value.
*Sensitivity* → structure containing information for the influence of uncertain elements on the performance margin.

To analyze the robust performance with the command *robustperf* , the calculating is made based on structured singular value bounds determined for normalized uncertainty $\left| \Delta(s) \right|_\infty \leq 1$

In conclusion, the robust performance is achieved when *lower bound* > 1, is not achieved when *upper bound* <1 and finally, when *lower bound* < 1 and *upper bound* > 1 is not possible to get a conclusion about the robust performance of the system.

### *1.2.9 Worst case gain*

The system performance should be evaluated approximately by the maximum of the frequency response of the largest singular value ( $H_\infty$ norm) of the sensitivity or complementary sensitivity transfer function. This largest value of the maximum represents the largest possible gain in the frequency domain, and this is defined as the "worst case" gain.

The command used to calculate this value is the following one, which has an input argument and three output arguments.

*[maxgain,wcunc,info] = wcgain(sys)*

Input argument:

*sys* → Represents the uncertainty model which is studying. It may be *uss* or *ufrd* class.

Output arguments:

*maxgain* → is a structure with three elements:
- *LowerBound*: lower bound of the worst case gain.
- *UpperBound*: upper bound of the worst case gain.
- *CriticalFrequency*: frequency response corresponding with the *LowerBound*.

*wcunc* → structure which contains a combination of uncertain element whose values maximize the system gain.

*info* → this structure is formed of two elements:
- *Frequency:* it includes the frequency vector used in the uncertain system analysis.
- *Info:* structure containing information for the influence of the uncertain elements on the worst-case gain.

The analysis of the worst case gain may be done in the case of the continuous time model and in the discrete time model.

### *1.2.10 μ-Synthesis*

μ-synthesis is one of the most important techniques of the Robust Control Design. With the appropriate weighting functions, is possible to ensure the robust stability and robust performance of the closed-loop system.

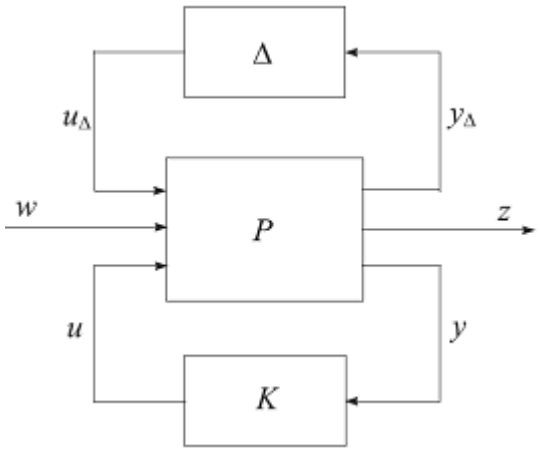



Figure 19: Block-diagram of the closed-loop system

Considering the block diagram above, which shows control problem in Linear Fractional Transformation, there are three blocks: $\Delta$, $P$ and $K$. The first one is the block who represents the uncertain element from the set $\Delta$ that parameterizes all supposed model uncertainty. $P$ block represents the known elements of the open-loop system, including nominal system mode, the performance weighting functions, as well as the uncertainty weighting functions. Finally, $K$ is the controller of the system. There are three input signal to the block P, $u_\Delta$ which is the uncertainty, $w$ for the references and disturbances and $u$ for controls. In case of the outputs, there are another three, $y_\Delta$ due to the uncertainties, $z$ for control outputs and errors and $y$ for measurements.
The set of systems to be controlled is described by the LFT:

$$\{F_u(\mathrm{P},\Delta): \Delta \in \mathbf{\Delta},\ \max_\omega \bar{\sigma}[\Delta(j\omega)] \le 1\} \tag{1.20}$$

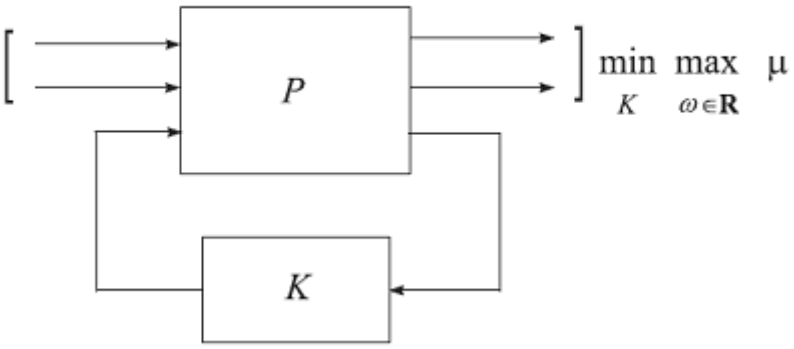
The aim of the design is to get the controller K which stabilize the system, at the same time that is closed-loop system is stable and is fulfilled $\Delta \in \mathbf{\Delta}$, $\max_\omega \bar{\sigma}[\Delta(j\omega)] \le 1$, satisfying:

$$\left\| F_u[F_L(P,K),\Delta] \right\|_\infty < 1 \tag{1.21}$$

Robust performance is get with K controller if and only if:

$$\mu_{\Delta P}(F_L(P,K)(j\omega)) < 1 \tag{1.22}$$

The objective of the μ-synthesis is to minimize the peak value of the structured singular value $\mu_{\Delta P}$ of the closed-loop transfer function matrix $F_L(P,K)$ over the set of all stabilizing controllers. The following expression and picture describe the aim of this technique:




Picture 20: μ-synthesis

$$\min_K \max_\omega \mu_{\Delta P}(F_L(P,K)(j\omega)) \quad K \text{ stabilizing} \tag{1.23}$$

*1.2.10.1 μ- synthesis by D-K iterations*

If it is wanted to get the controller who stabilizes the system in matlab, the command that is used is the following one:

*[k,clp,bnd] = dksyn(P,nmeas,ncont,op)*

Input arguments

*P* → Is the plant of the uncertain model which should be of class *uss* obtained with the command *sysic*.
*nmeas* → The number of plant measured inputs.
*ncont* → The number of plant control inputs.
*opt* → Is an optional object created by the command *opt = dktiopt* which has the elements below:

- *opt.FrequencyVector* – Vector with frequency values used in the μ-analysis. If not set, it is chosen automatically;
- *opt.InitialController* – Controller used to initiate first iteration, default is an empty SS object;
- *opt.AutoIter* - Automated mu-synthesis mode, default is 'on';
- *opt.DisplayWhileAutoIter* - Displays iteration progress in opt.AutoIter mode, default is 'off';
- *opt.StartingIterationNumber* - Starting iteration number, default is 1.
- *opt.NumberOfAutoIterations* - Number of D-K iterations to perform, default is 10;
- *opt.AutoScalingOrde*r - Maximum state order for fitting D-scaling data, default is 5;
- *opt.AutoIterSmartTerminat*e - Automatic termination of iteration procedure based on progress of design iteration, default is 'on';
- *opt.AutoIterSmartTerminateTol* - Tolerance used by *opt.AutoIterSmartTerminate*, default is 0.005.

Output arguments:

*K* → Designed controller (*ss* class)
*clp* → Closed-loop system model (*uss* object). It can be obtained also with the command *clp = lft(P,K)*.
*bnd* → Upper bound of the robust performance corresponding to the closed-loop system *clp*.
*Dkinfo* → N-by-1 cell array where N is the total number of iterations performed. The i cell contains relevant information like to following one:

- *K*: Controller at i-th iteration, ss object.
- *Bnds*: Robust performance bound for the closed-loop system.
- *MussvBnds*: Upper and lower μ bounds, an *frd* object computed by the auxiliary function *mussv*.
- *MussvInfo*: Structure returned from the function *mussv* at each iteration.

If it is wanted to know the controller in and exact iteration like the 5th one for example, the command that has to be tipped is:

*K_5 = dkinfo{5}.K*

In this case, the command *dksyn* can be used with continuous time model and with discrete time model also.

### *1.2.10.2 Practical aspects of μ-analysis and μ-synthesis*

Before starting with the development of robust control, is advisable to follow some advises to make the process easier:

1. Is advisable to begin the process with a simplified uncertain system due to the efforts that suppose the derivation of the uncertainty plant model and the complication of the controller. Once that those requirements are satisfied, it is appropriate to became the system more complex.
2. It is necessary to be cautious with the introduction of many sources of uncertainties, disturbances and noises due to the usage of μ, which suppose the worst-case gain. In that case, the development becomes less and less possible for the worst-case to appear.
3. There are always uncertainties for the inputs and outputs, so is advisable to use the relative (multiplicative) uncertainty.
4. In that case in which is get the value of μ bigger than 1, this could be due to the high requirements that are asking to the system. In those cases, is advisable to loosen the requirements and change the weighting function, with the objective of getting results below to 1, but near.
5. n the case that is wanted to make the system in discrete time, is preferable first to make that system in continuous time model due to the reason that this one is easier to get, and once that this one is get, then it would be easier to get the result in discrete time model.

# Chapter 2: Quad-rotor model

As it was explained in the beginning of the work, the device that is going to be used for the development of this work is a quad-rotor, included inside of the UAV (Unnamed Aerial Vehicles). His structure and dynamics are easier than conventional helicopters and it has less control complexity also. However, the quad-rotor is an unstable platform and impossible to fly in full open-loop system. Before starting with explanation of the model, in the table below is shown the notation that is going to be used during all the thesis.

## 2.1 Notation:

| *Symbol* | *Meaning* | *Unit* |
|---|---|---|
| $\phi,\theta,\psi$ | Roll, pitch and yaw angles | rad |
| u,v,w | Longitudinal, lateral and normal velocities | m/s |
| p,q,r | Roll, pitch and yaw rates | rad/s |
| x,y z | Position coordinates in North-East-Down (NED) frame | m |
| X,Y,Z | Forces applied along X, Y and Z body frame axes | N |
| L,M,N | Moments about X, Y and Z body frame axes | N m |
| $\Omega_i$ | Angular velocity of the i-th rotor | rad/s |
| $Q_i$ | Torque of the i-th rotor | N m |
| $T_i$ | Thrust of the i-th rotor | N |
| $I_{rotor}$ | Total main rotor moment of inertia | Kg $m^2$ |
| $V_x, V_y, V_z$ | Velocities in NED frame | m/s |
| $I_{xx}, I_{yy}, I_{ZZ}$ | Rolling, pitching and yawing moments of inertia | kg $m^2$ |
| d | Distance between rotor and centre of gravity of the quad-rotor | m |
| $U_{red\_i}$ | Control action of the i-th rotor | Volts |
| g | Acceleration of the gravity | m/$s^2$ |
| m | Total mass of the quad-rotor | kg |
| $u_{wind}, v_{wind}, w_{wind}$ | Wind velocities along the X, Y and Z body frame axes | m/s |

Table 1: Notation

This device is formed of four rotors which rotate at a very high rotational speed. These rotors are situated in the ends of two arms which are forming a cross. Two of them rotate in the clockwise direction and the two others in opposite direction with the aim of the cancellation of the yaw moment. The control forces are created varying the velocity of the different rotors. In the table below there is a little explanation of how the control action changes depending on the variation of the velocities of rotors.

| Force/Moment | $\Omega_1$ | $\Omega_2$ | $\Omega_3$ | $\Omega_4$ |
|---|---|---|---|---|
| Roll moment | | - | | + |
| Pitch moment | + | | - | |
| Yaw moment | + | - | + | - |
| Vertical thrust | + | + | + | + |

Table 2: Relation between angular velocities and movements of the quad-rotor

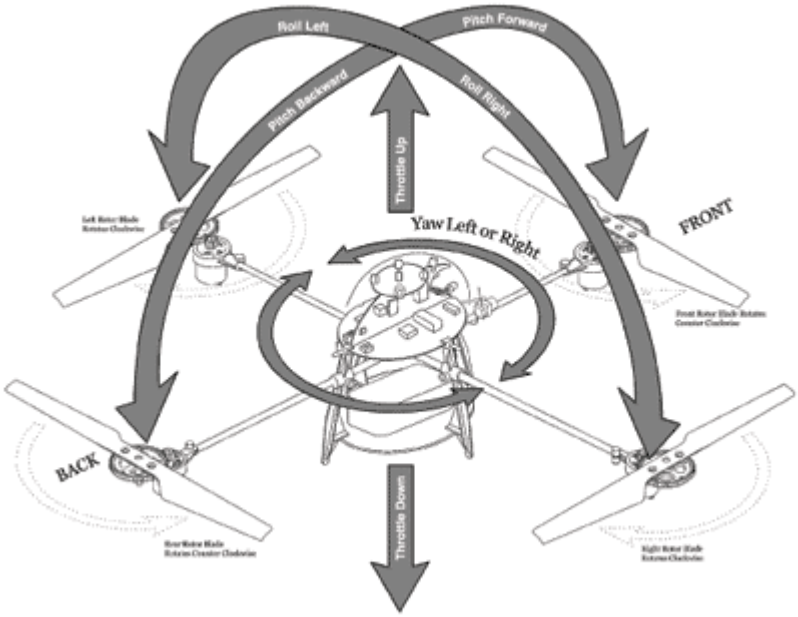



Figure 21: Scheme of the quad-rotor and his movements

In Table 2 is shown how are created different movements with the variation of the velocities of the four rotors. The + sign means that for getting that movement is necessary to increase the speed of the rotor, and in case of – sign is the opposite, the speed has to be decreased for getting the moment.

## *2.2 Quad-rotor nonlinear model*

### *2.2.1 Kinematics model*

Like it was explained in the previous part, the device is formed by 4 rotors situated in the end of two arms which are joined in the centre forming a cross. In one hand there is body frame, will have one orientation which is described respect *x*, *y* and *z* axis. In the other hand, there is another coordinates which are represented respect to the ground, they are described with *N*, *E* and *D* letters (North, East and Down). These two different axis systems are represented in the figure below.

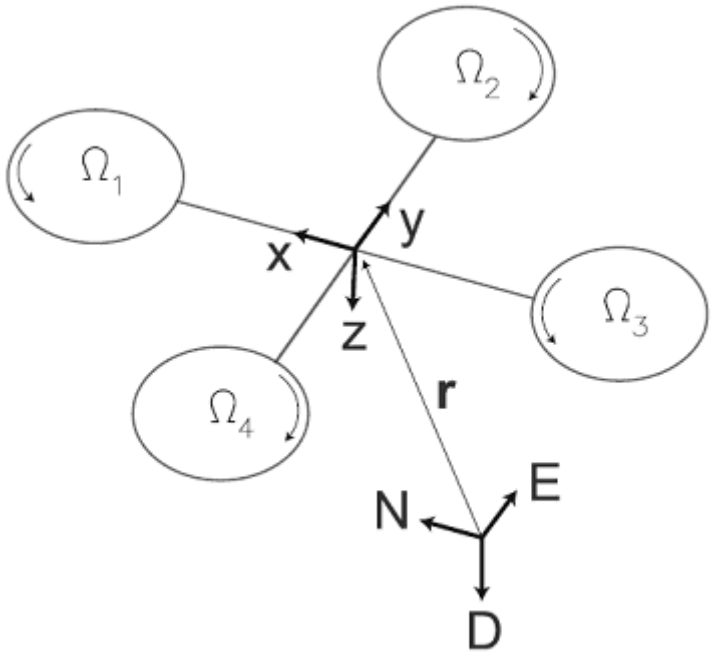



Figure 22: Scheme of the quad-rotors frames

Due to the reason that there are two different axis systems, is necessary to define an axis transformation matrix. If is wanted to transform the coordinates from the body frame to inertial frame, it can de done in more than one way, the following one is one possibility of the transformation matrix.

$$R = \begin{bmatrix} c\psi * c\theta & c\psi * s\phi - c\phi * s\psi & s\phi * s\psi + c\phi * c\psi * s\theta \\ c\theta * s\psi & c\phi * c\varphi + s\phi * s\psi & s\theta * s\varphi * s\theta - c\psi * s\phi \\ -s\theta & c\theta * s\phi & c\phi * c\theta \end{bmatrix} \tag{2.1}$$

Where *c* represent the *cos* of the angle and *s* represent the *sin* of the angle.
The order of rotation that is used for getting this matrix in first place is rotation of yaw angle $\psi$ respect *z* axis, in second place is the rotation of the pitch angle $\theta$ respect to the *y* axis and finally the rotation of roll angle $\phi$ respect to the *x* axis.
The Euler rates $\dot{\eta} = [\dot{\phi} \quad \dot{\theta} \quad \dot{\psi}]^T$ and angular body rates $\omega = [p \quad q \quad r]^T$ are related with the next expression:

$$\omega = R_r * \dot{\eta} \tag{2.2}$$

Where:

$$R_r = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi * \cos\theta \\ 0 & -\sin\phi & \cos\phi * \cos\theta \end{bmatrix} \tag{2.3}$$

In the case of the hovering position it is assumed that $R_r \approx I_{3x3}$, where I represents the identity matrix.

Even so, in Matlab model is going to be used the opposite relation between angular body rates and Euler rates. Thus, new equations will be the following:

$$\dot{\eta} = R_r^{-1} * \omega \tag{2.4}$$

$$\dot{\phi} = p + \sin(\phi)\tan(\theta)q + \cos(\theta)\tan(\theta)r \tag{2.5}$$
$$\dot{\theta} = \cos(\phi)q - \sin(\phi)r \tag{2.6}$$
$$\dot{\varphi} = (\sin(\phi)/\cos(\theta))q + (\cos(\phi)/\cos(\theta))r \tag{2.7}$$

### *2.2.2 Dynamics model*

The dynamics model is composed of a rotational movement and translation movement. The translational motion is underactuated while the rotational motion is fully actuated.
In the case of the translational equations, on the lines below are represented the translatory velocities in the body frame which are got beginning from the Newton's Second Law:

$$\dot{u} = vr - wq - g\sin(\theta) + X/m \tag{2.8}$$
$$\dot{v} = wp - ur + g\sin(\phi)\cos(\theta) + Y/m \tag{2.9}$$
$$\dot{w} = uq - vp + g\cos(\phi)\cos(\theta) + Z/m \tag{2.10}$$

Where X, Y and Z are forces applied along the X, Y and Z body frame axes.

In the case of the rotational equations, they are derived in the body frame using the Newton-Euler method:

$$\dot{p} = qr(I_{yy} - I_{zz})/I_{xx} + L/I_{xx}\,\mathrm{I} \tag{2.11}$$
$$\dot{q} = pr(I_{zz} - I_{xx})/I_{yy} + M/I_{yy} \tag{2.12}$$
$$\dot{r} = pq(I_{xx} - I_{yy})/I_{zz} + N/I_{zz} \tag{2.13}$$

Where L, M and N are Moments about the X, Y and Z body frame axes. Which are got in the form below:

$$L = -I_{rotor}\,q(\Omega_1 - \Omega_2 + \Omega_3 - \Omega_4) + d*(-T_2 + T_4) \tag{2.14}$$
$$M = I_{rotor}\,p(\Omega_1 - \Omega_2 + \Omega_3 - \Omega_4) + d*(T_1 - T_3) \tag{2.15}$$
$$N = (-Q_1 + Q_2 - Q_3 + Q_4) \tag{2.16}$$

Where $I_{rotor}$ is the total main rotor moment of inertia, $d$ is is distance between the rotor and centre of gravity of the quad-rotor, $T_i$ is the thrust generated buy the i-th rotor and $Q_i$ is the torque generated by the i-th rotor.

In this papers a simplification will be made in order to get a simpler relation between the angular velocities and rotors thrust and torque, because actually the relation between them depends of some different elements like density of the air, radius of the shape, pitch angle of the blade and other factors. For getting thrust and torque, it is necessary to do some experiments with the rotors in order to get these relations between the angular velocity of the rotors and thrust and torque.
For the development of this project, the following relation will be used:

$$T_i = 3.13*10\text{^}(-5)*\Omega_i^{\,2} \tag{2.17}$$

$$Q_i = 7.5*10\text{^}(-7)*\Omega_i^{\,2} \tag{2.18}$$

In the other and, for the development of these previous equations, it is necessary to know how the angular velocity is calculated. In this case it will be a function of the input control reference signal and a constant. Thus, angular velocities equation expression will be the next one:

$$\Omega_i = 5.389*U_{ref\_i} \tag{2.19}$$

For the control of the velocity and position it will be necessary to know also the relationship between the velocities in body frame and in the Earth fixed frame. For this aim is used the matrix R shown at the beginning of this chapter in the following way:

$$\begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = \begin{bmatrix} c\psi*c\theta & c\psi*s\phi - c\phi*s\psi & s\phi*s\psi + c\phi*c\psi*s\theta \\ c\theta*s\psi & c\phi*c\varphi + s\phi*s\psi & s\theta*s\varphi*s\theta - c\psi*s\phi \\ -s\theta & c\theta*s\phi & c\phi*c\theta \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{2.20}$$

## *2.3 Simulink model*

For the simulation of the quad-rotor a Simulink model is used, due the facilities that this toolbox provides. In this case, inside the Simulink model there is a toolbox which corresponds with the theme of Unnamed Aerial Vehicles which has different models for representing different kind of these devices. Even so, in this case the model of the quad-rotor is going to be defined with and S-function which is defined in Matlab.

In the case of this thesis, the Simulink model is given due to the fact that it was done a time ago. So, for the development of this project, some changes will be made in different elements of the model wit the aim of getting the best μ-synthesis controller with the most near behaviour of the model to the reality.

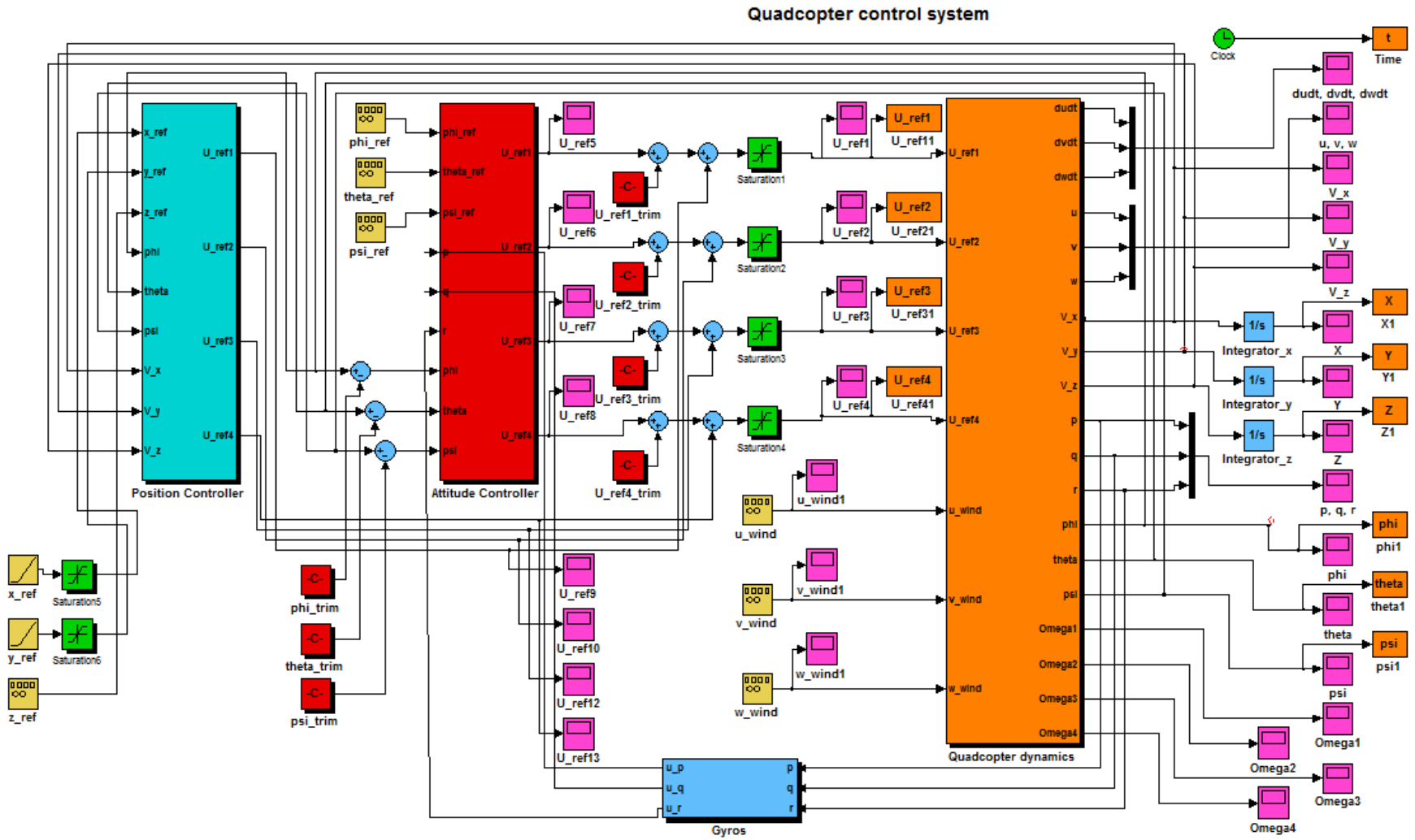



Figure 23: Simulink model of the quad-rotor plant with controllers

As if it is shown in the figure 23, in the model there are two controllers the first one represents position controller which some *x*, *y* and *z* desired position are introduced with some signals generators and also the actual quad-rotor positions introduced. With these data, controller calculates control signals that have to be introduced in the quad-rotor dynamics model for getting the desired position. The second one represents the quad-rotor attitude controller in which the trim values of angles are introduced with the actual attitude data and controller calculates control signals that have to be introduced in the quad-rotor dynamics model in order to get the desired attitude.

In the right part with orange rectangle are represented dynamics of the quad-rotor, which inside has an S-function for the development of the computations for the desired operation of it. In this function some inputs will be introduced as control signals needed for getting desired behaviour and also some signals which will be used for representing wind disturbances. In the other side of the rectangle outputs are got, nineteen in this case, which some of them are sent to the controllers as feedbacks in order to get the computations that in paragraphs above were are explained. All the output signals are represented in a scope with the objective of knowing the behaviour of the device.

There is also another function called *Gyros* which is used to introduce some delays in measurement due to the dynamics that should be considered in the gyroscope

## *2.4 Linearized model*

Once that the Simulink model is described, the next step that is has to be taken is the linearization of the model. For that aim some commands and state-space will be explained in the following lines.

In the case of the quad-rotor model, as in most of the cases, the state-space model is defined with the following equations, equations that were described in the previous lines:

$$\dot{x} = Ax + Bu \tag{2.21}$$
$$y = Cx + Du \tag{2.22}$$

Where:

- $x$ is the state vector with $n$ dimension ;
- $y$ is the output vector with $q$ dimension;
- $u$ is the control or input vector with $p$ dimension;
- $A$ is the matrix of the system with $nxn$ dimension;
- $B$ is the input matrix with $nxp$ dimension;
- $C$ is the output matrix with $qxn$ dimension;
- $D$ is the feedforward matrix with $qxp$ dimension;

For the quad-rotor:

$$x = \{u, v, w, p, q, r, \phi, \theta, \psi\}$$
$$u = \{U\_ref1, U\_ref2, U\_ref3, U\_ref4, M\_dx, M\_dy, M\_dz\}$$
$$y = \{\dot{u}, \dot{v}, \dot{w}, u, v, w, V\_x, V\_y, V\_z, p, q, r, \phi, \theta, \psi, \Omega_1, \Omega_2, \Omega_3, \Omega_4\}.$$

In one hand, linearized model is got using the Simulink model described in the figure below and using *linmod* function in Matlab, which gets directly the linearization of the model that is studying. This routine is used in *num_lin_quad* function shown in the last part of the thesis, where all Matlab functions are described. As can be seen in the picture, the model is formed of seven inputs (four control actions and three wind disturbances) and sixteen outputs.

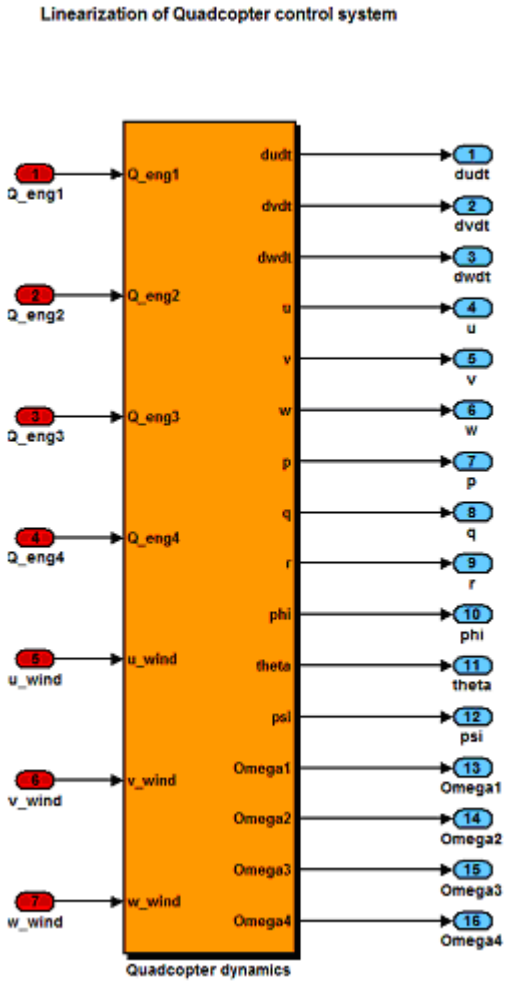



Figure 24: Simulink model of the linearization of the quad-rotor

If it is wanted to know A, B, C and D matrices of the linearized system, in the following lines commands that should be tipped and the results are shown:

- G_quad.a

$$A = \begin{bmatrix} -0.0000 & 0.0000 & 0.0000 & 0 & 0 & 0 & 0 & -9.8066 & 0 \\ -0.0000 & -0.0000 & 0.0000 & 0 & 0 & 0 & 9.8066 & 0 & 0 \\ 0 & 0 & -0.0000 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.0000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.0000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0000 & 0 & 0.0000 & 0 & 0.0000 & 0 \\ 0 & 0 & 0 & 0 & 1.0000 & -0.0000 & -0.0000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.0000 & 1.0000 & -0.0000 & 0 & 0 \end{bmatrix}$$

- G_quad.b

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.1335 & -0.1335 & -0.1335 & -0.1335 & 0 & 0 & 0 \\ 0.0000 & -1.0357 & 0.0000 & 1.0357 & 59.6659 & 0 & 0 \\ 1.0357 & -0.0000 & -1.0357 & -0.0000 & 0 & 59.6659 & 0 \\ -0.0691 & 0.0691 & -0.0691 & 0.0691 & 0 & 0 & 43.2152 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- G_quad.c

$$C = \begin{bmatrix} -0.0000 & 0.0000 & 0.0000 & 0 & 0 & 0 & 0 & -9.8066 & 0 \\ -0.0000 & -0.0000 & 0.0000 & 0 & 0 & 0 & 9.8066 & 0 & 0 \\ 0 & 0 & -0.0000 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1.0000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.0000 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.0000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.0000 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

G_quad.d

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.1335 & -0.1335 & -0.1335 & -0.1335 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5.3890 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5.3890 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5.3890 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5.3890 & 0 & 0 & 0 \end{bmatrix}$$

In the other hand, it is necessary to introduce another Matlab function called *trim*, which try to find the trim values of the state and input vectors components from the nonlinear quad-rotor model. Trim point is an equilibrium point that in the parameter space of a dynamic system at which the system is in a steady state. In the case of aircrafts is a setting of its controls that causes the aircraft to fly straight and level. Mathematically, a trim point is a state in which system state derivates are equal to zero. In the case that the trim value can not be found the functions returns the point that minimizes the maximum deviation from zero of the derivates.

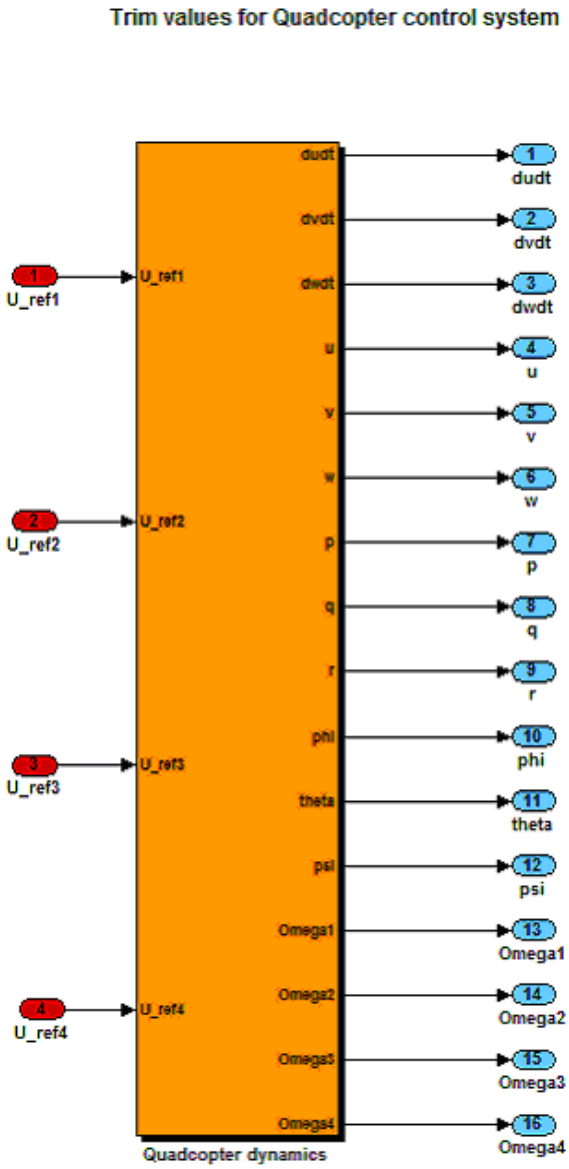



Figure 25: Simulink model for the calculation of the trim values

As it was explained in the first chapter, when in Robust Control a model is going to be defined, is necessary to introduce uncertainties which could be of two different cases: multiplicative and additive.

The first one is that one that is chosen for representing the simplifications that are taken and for approximation errors. In these cases an uncertainty of 10% is taken for each control signal, getting the following model:

$$G(s) = G_{quad}\begin{bmatrix} 1+\delta_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1+\delta_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1+\delta_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1+\delta_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.23}$$

Where like it is said in the paragraph above: $|\delta_i| \leq 0.1$.

Thus, the uncertain model of the quad-rotor that is got after the previous developments is:

$$y = G(s)u \tag{2.24}$$

# Chapter 3: Controller design

In this thesis two controllers are going to be used as it was explained in the previous chapter. One is a PD controller which is used to get the control of the altitude of the quad-rotor. The other one is an attitude controller which it got with μ-synthesis in order to ensure stability and robustness. The controller will ensure small deviation of the Euler angles from the trim values in presence of wind disturbances.

## 3.1 Attitude controller

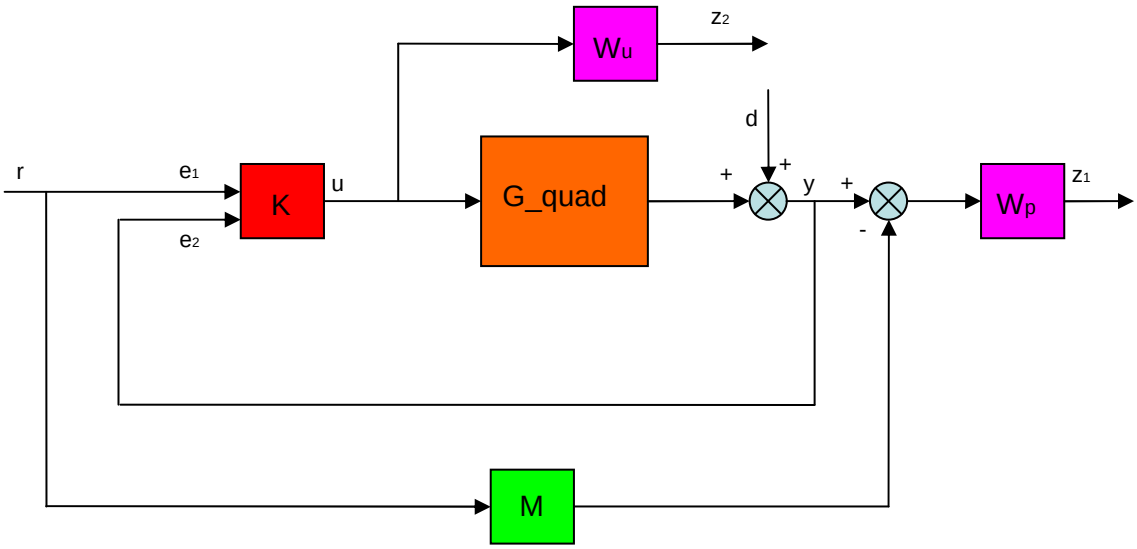



Figure 26: Block diagram of the closed-loop system

In the μ-synthesis, for obtaining good performance in the closed-loop system, two-degree-of-freedom controller should be implemented, as it is shown in the figure below. In this case of the two-degree-of-freedom controller, control actions are generated in the following way:

$$u_c = \begin{bmatrix} K_r & K_c \end{bmatrix} \begin{bmatrix} r_c \\ y_c \end{bmatrix} = K_r r_c + K_c y_c \tag{3.1}$$

Where $r_c$ is the reference vector, $y_c$ is the output feedback vector, $K_r$ is the pre-filter transfer function matrix and $K_c$ is the output feedback transfer function matrix. The first two elements are the vectors below:

$$r_c = \begin{bmatrix} \phi_{ref} & \theta_{ref} & \psi_{ref} \end{bmatrix}^T \tag{3.2}$$

$$y_c = \begin{bmatrix} \phi & \theta & \psi & p & q & r \end{bmatrix}^T \tag{3.3}$$

M is the ideal dynamic model that the designed closed-loop system should match to and $e_y$ and $e_u$ are the outputs vector signals, which they should fulfil the following relation:

$$\begin{bmatrix} e_y \\ e_u \end{bmatrix} = \begin{bmatrix} W_p(S_0 G_u W_s K_r - M) & W_p S_0 G_d \\ W_u S_i K_r & -W_u S_i K_y G_d \end{bmatrix} \begin{bmatrix} r_c \\ d \end{bmatrix} \tag{3.4}$$

Where $S_i$ and $S_o$ are the input and output sensitivity matrices. Which are calculated in the form below:

$$S_i = (I + K_y G_u W_s)^{-1} \tag{3.5}$$

$$S_o = (I + G_u W_s K_y)^{-1} \tag{3.6}$$

In one hand, for all the uncertain plants model of G, the performance criterion requires that the transfer function matrix of the exogenous signal ( $r_c$ and d) to the output signal vector explained previously, have to be small in the sense of $\|\cdot\|_\infty$.

In the other hand, $W_u$ and $W_p$ are used for taking account of the relative importance of the of different frequency ranges for which the performance requirements should be fulfilled.

In the following table are explained the four function matrices that are used to relate the exogenous signals and output signal.

| Function | Description |
|---|---|
| $W_p(S_0 G_u W_s K_r - M)$ | Weighted difference between real in ideal closed-loop system |
| $W_p S_0 G_d$ | Weighted sensitivity to disturbance |
| $W_u S_i K_r$ | Weighted control action due to reference |
| $-W_u S_i K_y G_d$ | Weighted control action due to disturbance |

Table 3: Functions of matrix to be minimized

As it is explained in the first chapter, the goal of the design is to get robust stability and robust performance of the closed-loop system in presence of plant uncertainties and output disturbances. In the following two lines are reminded both concepts:

- Robust stability: is got when the system is internally stable for all the plants of the model.
- Robust performance: the condition below should stay for al G and in addition the following performance criterion should be satisfied.

In order to get robust performance the following inequality has to be fulfilled:

$$\left\| \begin{bmatrix} W_p(S_0 G_u W_s K_r - M) & W_p S_0 G_d \\ W_u S_i K_r & -W_u S_i K_y G_d \end{bmatrix} \right\|_\infty < 1 \tag{3.7}$$

In this case, the main aim of the attitude controller of the quad-rotor is to keep the angular velocities of $\phi$, θ and ψ close to reference values in presence of wind disturbances and measurement noises.

In order calculating the controller 9 inputs are need: three reference angles, angles that quad-rotor has when it is flying and also their rates. For these last six variables is necessary to have a really good feedback, but always taking account that the quad-rotors translational motion is slower than the dynamics of its angular motion.

In the design of the controller, the transfer function matrix M of the ideal matching model as diagonal is chosen in order to suppress the interaction between the three channels and is taken as

$$M(s) = \begin{bmatrix} w_{m1} & 0 & 0 \\ 0 & w_{m2} & 0 \\ 0 & 0 & w_{m3} \end{bmatrix} \tag{3.8}$$

where

$$w_{m1} = w_{m2} = w_{m3} = \frac{1}{3s^2 + 4.2s + 1} \tag{3.9}$$

In this thesis, the μ-synthesis for four different cases of performance and control weighting functions is done. Bases of both matrixes are the same, only the gain of the transfer functions is changed in the different cases. These following matrixes are the bases of the weighting functions.

$$W_p(s) = \begin{bmatrix} k_{p1}\dfrac{0.01s+1}{0.01s+0.1} & 0 & 0 \\ 0 & k_{p2}\dfrac{0.01s+1}{0.01s+0.1} & 0 \\ 0 & 0 & k_{p3}\dfrac{0.01s+1}{0.01s+0.1} \end{bmatrix} \tag{3.10}$$

And

$$W_u(s) = \begin{bmatrix} k_{u1}\dfrac{0.002s+1}{0.000003s+1} & 0 & 0 & 0 \\ 0 & k_{u2}\dfrac{0.002s+1}{0.000003s+1} & 0 & 0 \\ 0 & 0 & k_{u3}\dfrac{0.002s+1}{0.000003s+1} & 0 \\ 0 & 0 & 0 & k_{u4}\dfrac{0.002s+1}{0.000003s+1} \end{bmatrix} \tag{3.11}$$

Where $k_{pi}$ and $k_{ui}$ are gains of i-th performance and control weighting functions.

It is necessary to remind that the performance weighting functions are chosen as low pass filters to suppress the difference between the system and model, and the control weighting functions are chosen as high pass filters with appropriate bandwidth in order to impose constraints on the spectrum of the control actions.

Using these weighting functions, if values of μ near to 1 are got, it means that the robust performance is got. But, if the value of μ is high, this is due to the reason that the weighting functions are quite strict and if the value of μ is near to 0 it means that weighting functions are lax.

But, with the value of μ only the information that can be got is the fact that the system is strict or not and if robust stability and performance are got, but for realizing which weighting function can be changed for getting better value of μ it is necessary to look at input and output sensitivity functions, in which plots are represented control and performance weighting functions.

The magnitude frequency response of the inverse weighting function $W_p^{-1}$ is represented with a discontinuous in the plot of output sensitivity function, in which output sensitivity ( $S_O$ ) for random values of different uncertainties are represented with continuous lines. In order to get the desired value of μ, the discontinuous line that is representing inverse performance weighting function has to be in all the frequency range below of the continuous lines that are representing random values of the output sensitivity function.

For the case of the magnitude frequency response of the inverse weighting function $W_u^{-1}$, it is represented with a discontinuous in the plot of input sensitivity function, in which input sensitivity ( $S_i$ ) for random values of different uncertainties are represented with continuous lines. In the same way as in the inverse performance weighting function, the discontinuous line has to be below the continuous lines in whole the frequency range.

## *3.2 Altitude controller*

The case of the altitude controller is less complicate than the case of the attitude controller. In this case a PD controller is used to get the desired position. In the function of the position controller 9 inputs are introduced (x_ref, y_ref, z_ref, phi, theta, psi, V_x, V_y, V_z) with the aim of getting 4 outputs which will be introduced in the four different rotors of the device.
In this controller three aspects are controlled: altitude and roll and pitch angles, and from this three parameters fours control signals are got in the following way:

```
Altitude control law
    K_p1 = -10.0;
    K_d1 = -100.0;
    U1 = (K_p1*(z_ref - z) + ...
                              K_d1*(z_ref_dot -
V_z))/(cos(phi)*cos(theta));
%
%   Roll control law
    K_p2 = 5.0;
    K_d2 = 50.0;
    U2 = (K_p2*(y_ref - y) + K_d2*(y_ref_dot - V_y))/cos(psi);
%
%   Pitch control law
    K_p3 = -5.0;
    K_d3 = -50.0;
    U3 = (K_p3*(x_ref - xx) + K_d3*(x_ref_dot -
V_x))/(cos(phi)*cos(psi));
%
%   Calculate outputs
    U_ref1 = U1/4 + U3/2;  % Motor references
    U_ref2 = U1/4 - U2/2;
    U_ref3 = U1/4 - U3/2;
    U_ref4 = U1/4 + U2/2;
```

As can be seen, there are 3 PD controllers, one per each variable that want be controlled.

In last 4 lines it is shown how are got reference signals.
For the control of the pitch angle, the velocity of the engines number 1 and 3 have to be changed to get the desired movement, and something similar with the engines 2 and 4 for getting the desired roll angle.

## *3.3 Cases of weighting functions*

In the following part of the chapter different cases of control and performance weighting functions that are used in order to get a good controller are shown. For that aim, as it is explained, input and output sensitive functions plot are studied. In all the cases, weighting functions are the same due to the reason that is wanted to have the same restriction in all the channels.

### *3.3.1 Case 1*

In the first case, the following values are taken for the gains of control and performance weighting functions.

- $k_{pi} = 1.3$
- $k_{ui} = 0.003$

Before applying μ-synthesis, the open-loop system has to be created taking account the first picture of this chapter. For that aim *sysic* command is used, as can be seen in the last pages where the Matlab functions are shown.

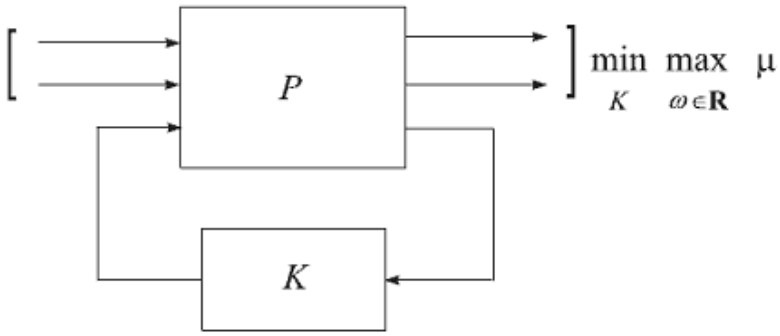



Figure 27: Block diagram of μ-synthesis

In the figure above μ-synthesis is represented, where P is the transfer function matrix of the extended loop system. With P, uncertain plant model, matching model and performance weighting functions are represented.

Plots below of inverse performance and inverse control weighting function are got after applying *olp_quad* function shown in final pages. Due to the reason that all of them they have the same value, in both cases only one line can be seen.

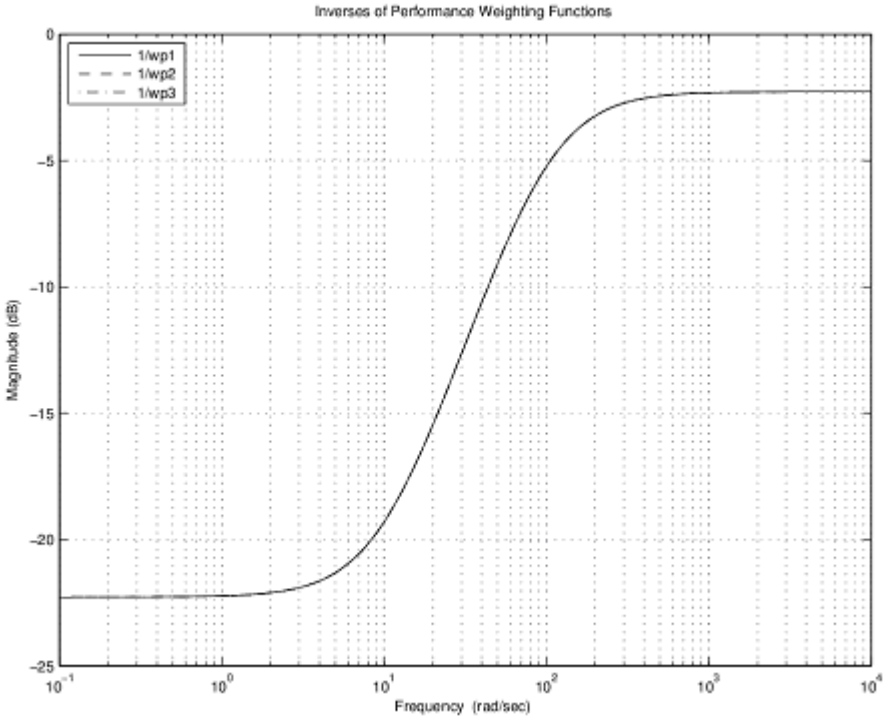



Figure 28: Inverse of Performance Weighting Functions
Case 1

Having a look in the plot, with the performance weighting function chosen for this first case, the noise that is introduced due to the feedback is suppressed more than 10 times (-23 dB).

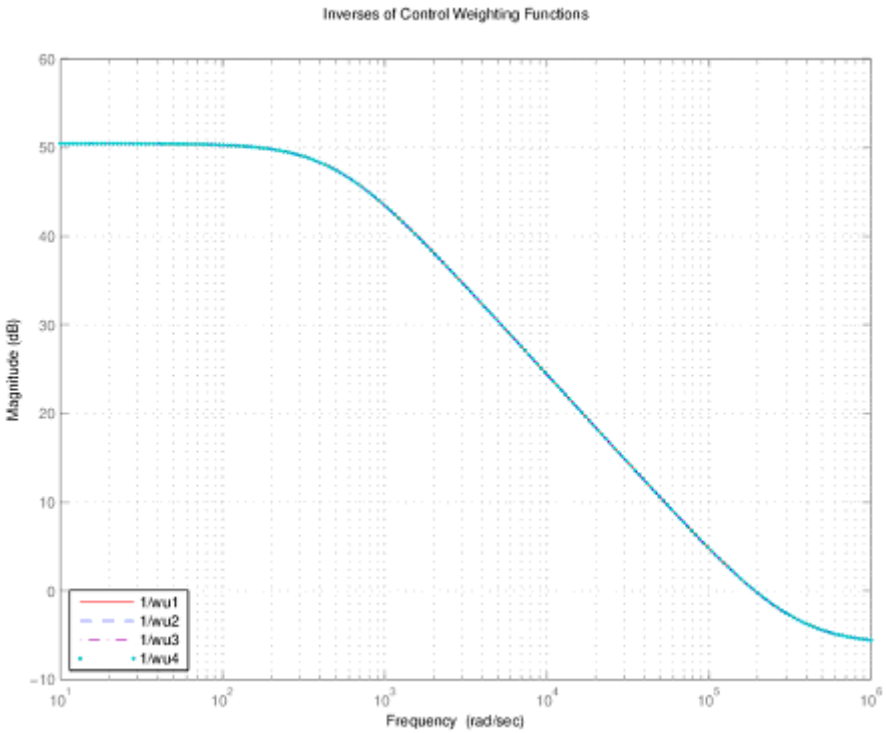



Figure 29: Inverse of Control Weighting Functions
Case 1

As can be seen in the second plot, control weighting functions which are used in order to suppress the disturbances which normally appear in low frequency range, are able to suppress disturbances 312 times (50 dB).

After the simulation of the *olp_quad* function, if the *ms_quad* function is used, which apply the continuous μ-synthesis by D-K iterations to the system with *dksyn* function, the results that are got are the following:

***Iteration Summary***

***----------***

| ***Iteration #*** | ***1*** | ***2*** | ***3*** | ***4*** |
|---|---|---|---|---|
| ***Controller Order*** | ***22*** | ***22*** | ***22*** | ***22*** |
| ***Total D-Scale Order*** | ***0*** | ***0*** | ***0*** | ***0*** |
| ***Gamma Acheived*** | ***1600.00*** | ***100.760*** | ***6.124*** | ***3.553*** |
| ***Peak mu-Value*** | ***98.773*** | ***2.991*** | ***1.736*** | ***1.671*** |

As can be seen, the order of the controller that is got 22, not so high if it is compared with other controllers that are used to get actually in other kind of devices. After 4 iterations, the peak value that is got for μ is 1.671, which is not so far from the objective of μ<1, but still, it can be improved.

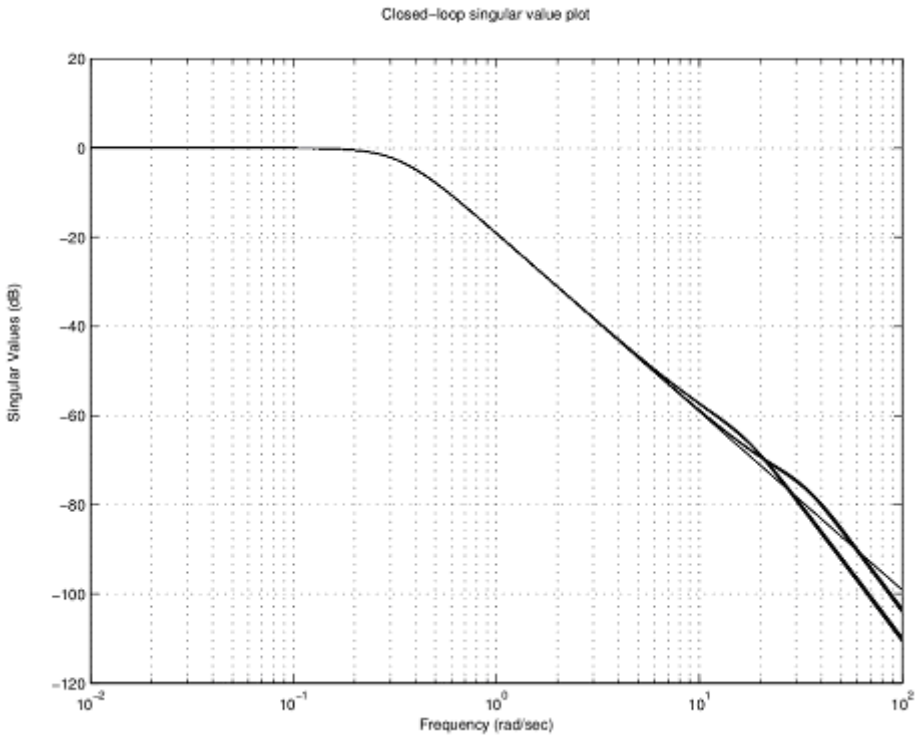



Figure 30: Singular value of the closed-loop system and desired response

Case 1

Plot above shows comparative between wanted system (M) and the closed-loop system that is got after applying μ-synthesis. As can be seen, during the frequency response two plots are more or less the same, besides at the end of the frequency response, where system that is got, that it is represented with more than one line due to the uncertainties, has a small deviation comparing with the wanted system.

Having a look on the plots about the input and output sensitivity response shown below, it is possible to see that discontinuous lines are below the continuous lines that represent the system for different values in both plots. That means that controller is really strict, so on the next cases gains of the control and performance weighting functions are changed in order to get less strict controller which fulfils the inequality $\mu<1$.

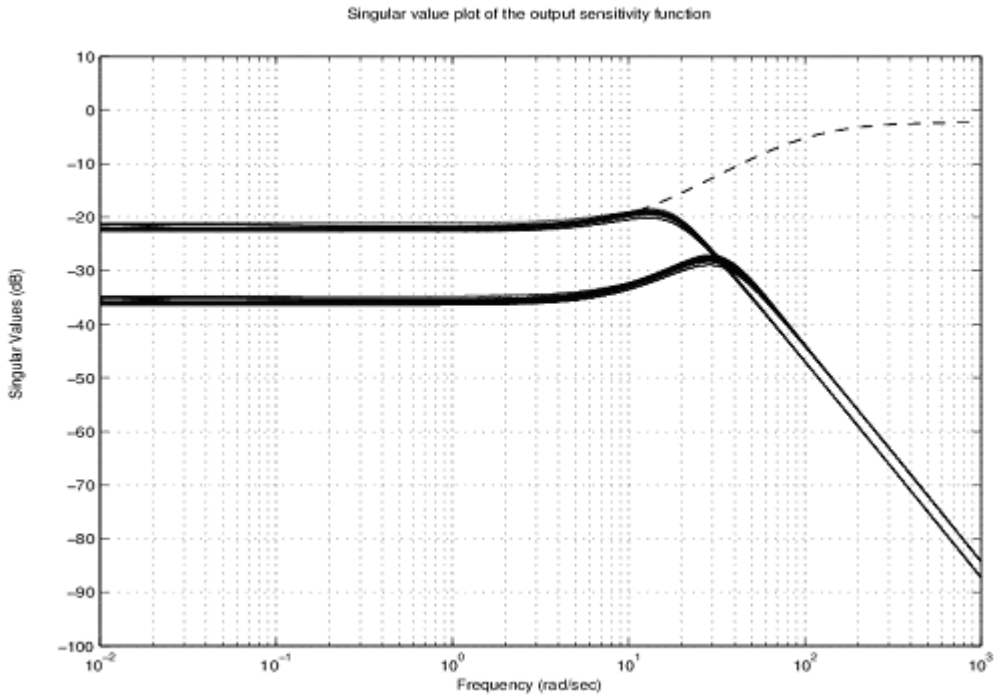



Figure 31: Output sensitivity function and inverse of Performance Weighting Function Case 1

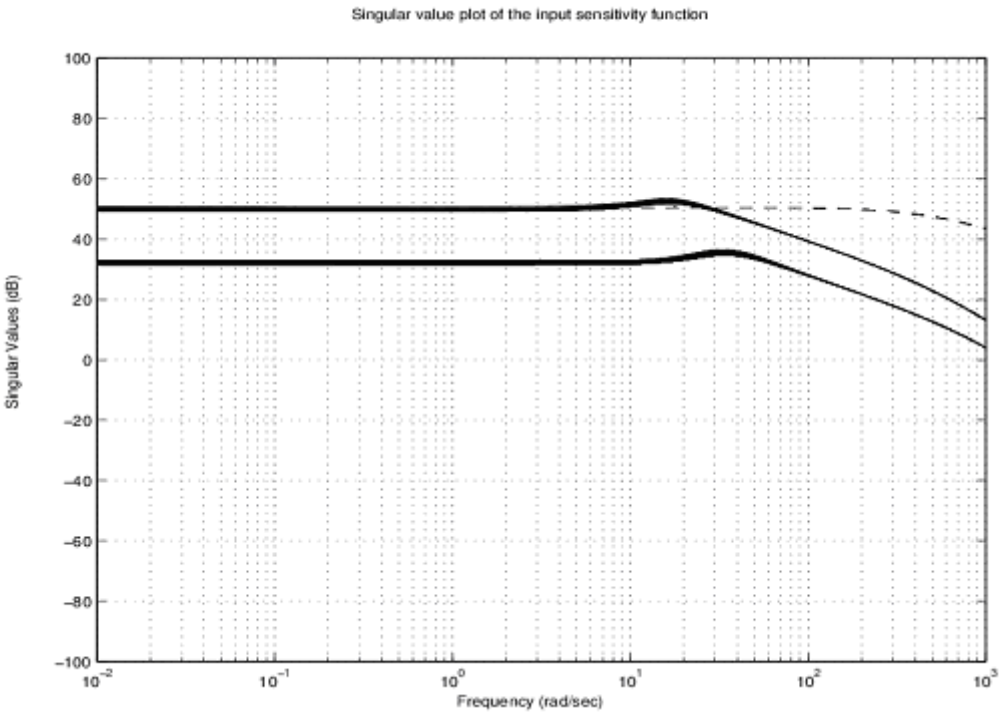



Figure 32: Input sensitivity function and inverse of Control Weighting Function Case 1

### *3.3.2 Case 2*

In order to get needs that are explained in the last paragraph of the first case, values of the gain of the weighting functions are decreased. In this way, discontinuous line of plots will be increased. For that, the following values are taken:

- $k_{pi} = 1.2$
- $k_{ui} = 0.002$

If it is applied the open-loop functions in the new system, plots that are got for the inverse performance weighting functions and for inverse control weighting functions are the following:

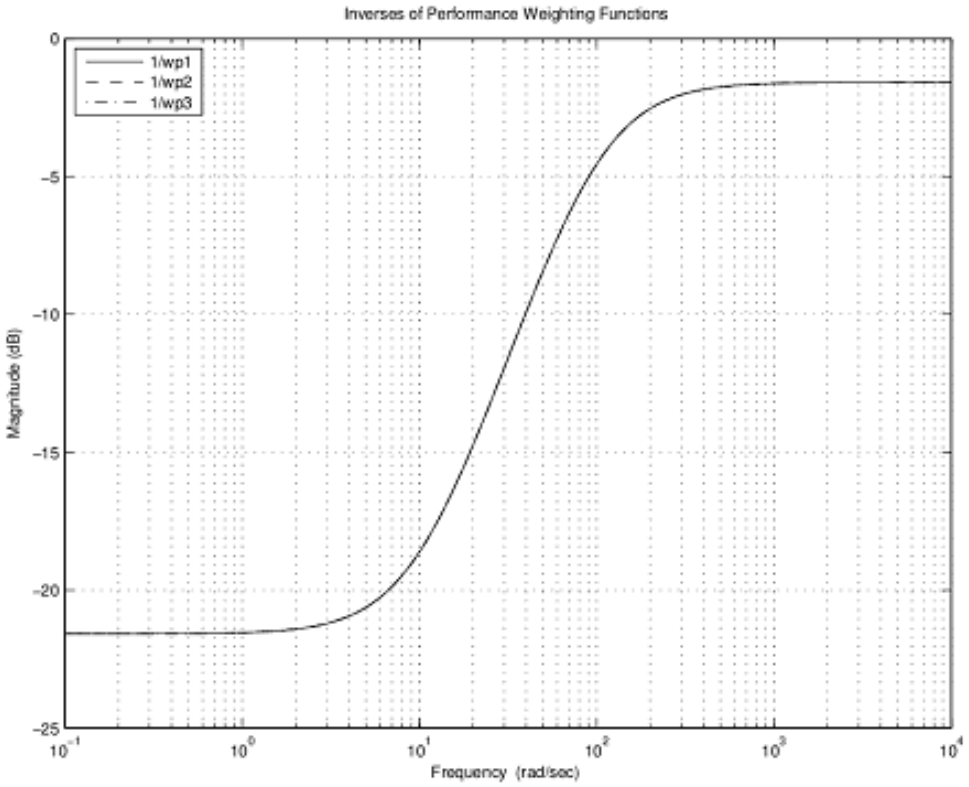



Figure 32: Inverse of Performance Weighting Functions Case 2

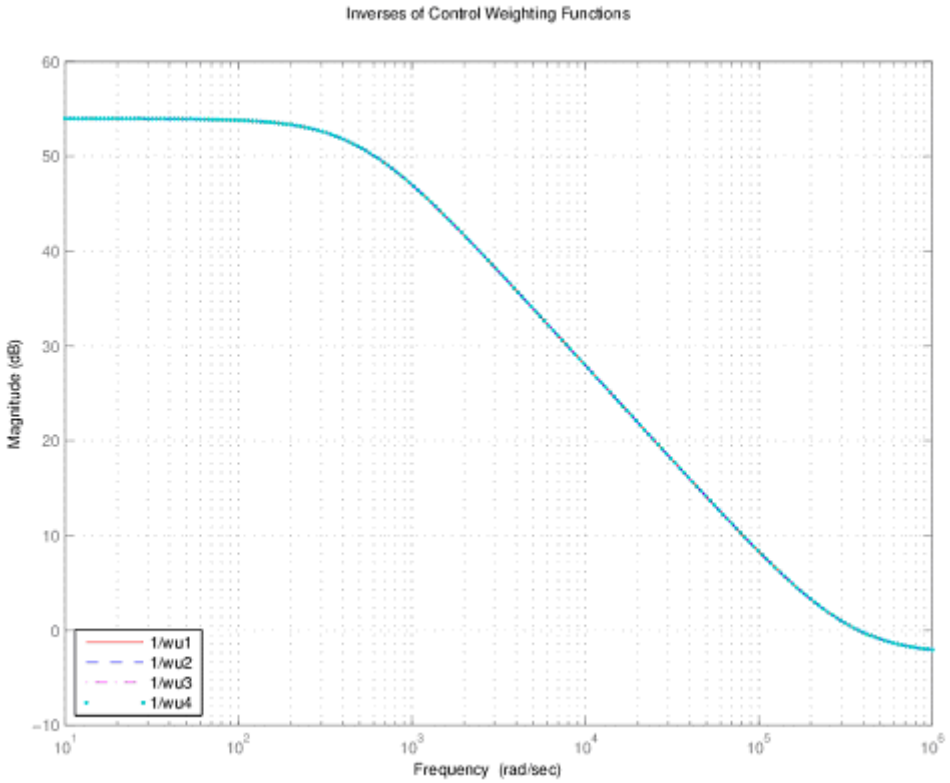



Figure 33: Inverse of Control Weighting Functions
Case 2

As can be seen in the both plots, curves start on a higher value, finish also in higher value and the form is the same; this is due to the fact that only the gain of the control weighting function is changed.

After μ-synthesis by D-K iterations is applied to the new system with the new weighting functions, results that are got for the controller are the following:

***Iteration Summary***

***----------***

| | | | | |
|---|---|---|---|---|
| ***Iteration #*** | ***1*** | ***2*** | ***3*** | ***4*** |
| ***Controller Order*** | ***22*** | ***22*** | ***22*** | ***22*** |
| ***Total D-Scale Order*** | ***0*** | ***0*** | ***0*** | ***0*** |
| ***Gamma Acheived*** | ***800.000*** | ***400.688*** | ***11.194*** | ***20.804*** |
| ***Peak mu-Value*** | ***98.197*** | ***2.781*** | ***1.277*** | ***1.252*** |

After 4 iterations the order of the controller that is get is 22, the same as in the first case, but the peak value for μ that is got is 1.252, quite better than that one of the first case but still can be improved, with some changes in weighting functions.

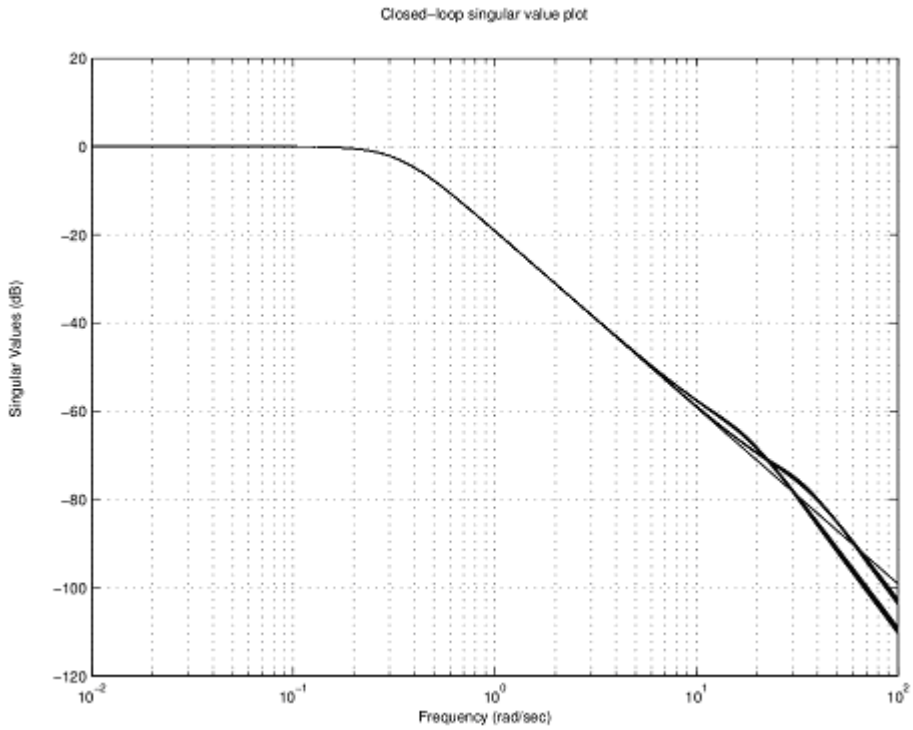



Figure 34: Singular value of the closed-loop system and desired response Case 2

In the same way like in the previous case, singular value of the closed-loop system is quite similar to the response of the desired system. Having a look in the plot of the closed-loop system of the first case and to this one not big difference can be appreciated.

In order to try to see difference between both cases, input and output sensitivity functions are going to be studied in the following lines.

As in the first case, taking a look in plots of output and input sensitivity functions, in the case of the first inverse performance weighting function is above the continuous lines that are representing the system with uncertainties. But it can not be said the same in the case of the input sensitivity function; where in most of the frequencies, control weighting function has higher value that the system, but there is a frequency range in which this fact is not fulfilled.

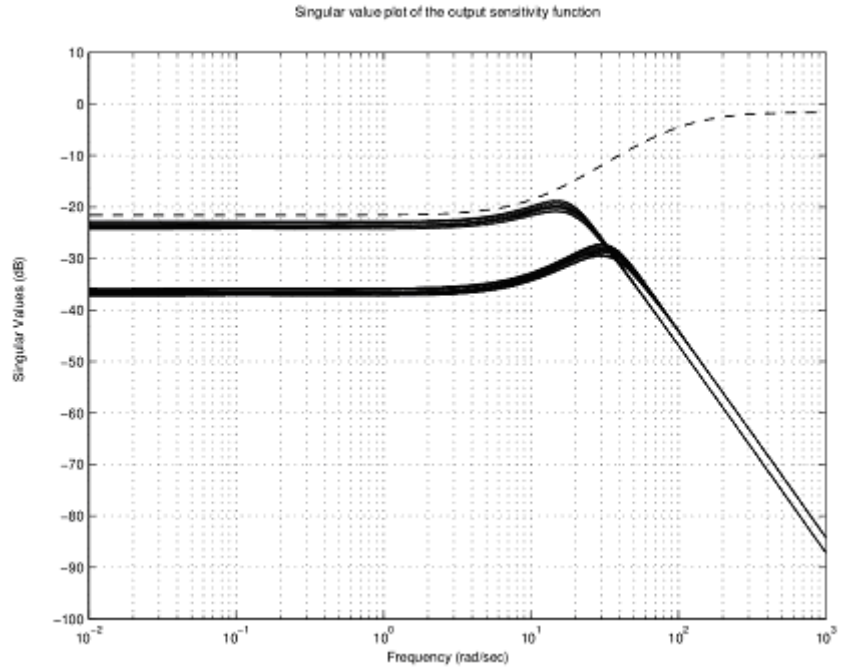



Figure 35: Output sensitivity function and inverse of Performance Weighting Function Case 2

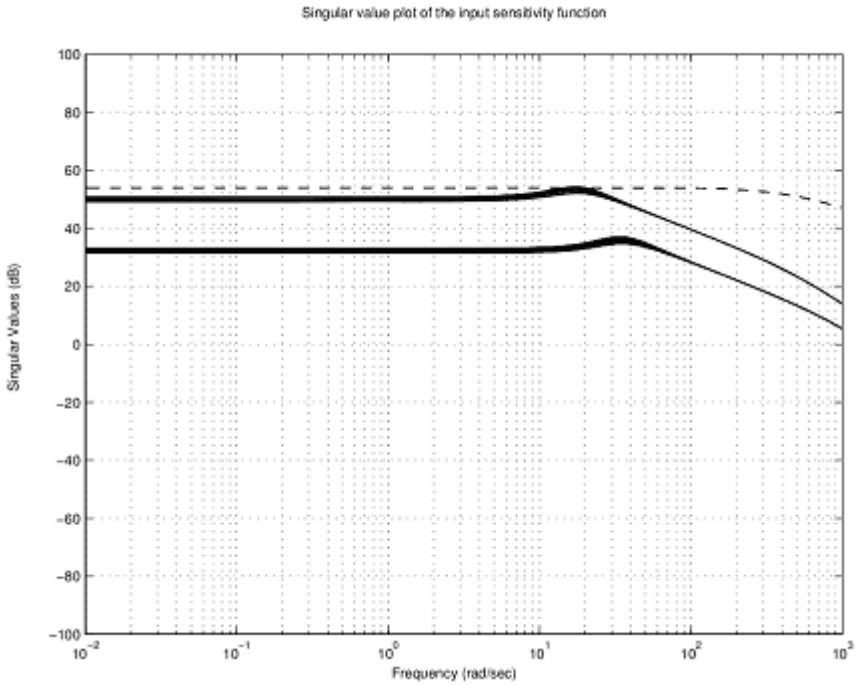



Figure 36: Input sensitivity function and inverse of Control Weighting Function Case 2

### *3.3.3 Case 3*

After the second case, the aim of getting a value of μ lower than 1 is not got, so in this third case values of the gains of control and performance are changed. Although that in the second case performance weighting function fulfils the conditions, in order to get higher margin, the value of the gain also it is changed. These are values taken for the weighting functions:

- $k_{pi} = 1.2795$
- $k_{ui} = 0.00185$

To follow with the routine of the previous cases, inverse control and performance weighting functions are shown. Changes that are made on gains produce the same movement explained in previous case.

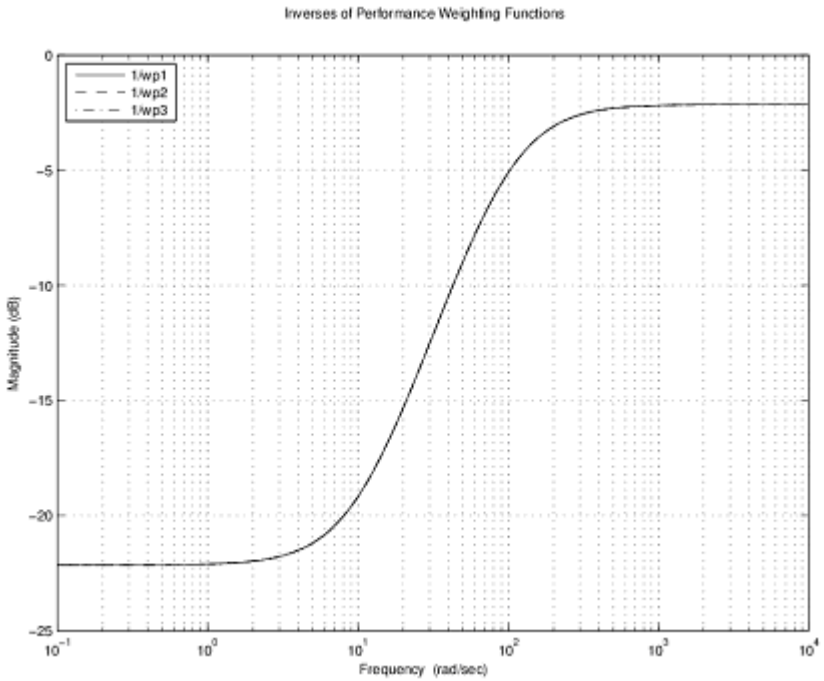



Figure 36: Inverse of Performance Weighting Functions Case 3

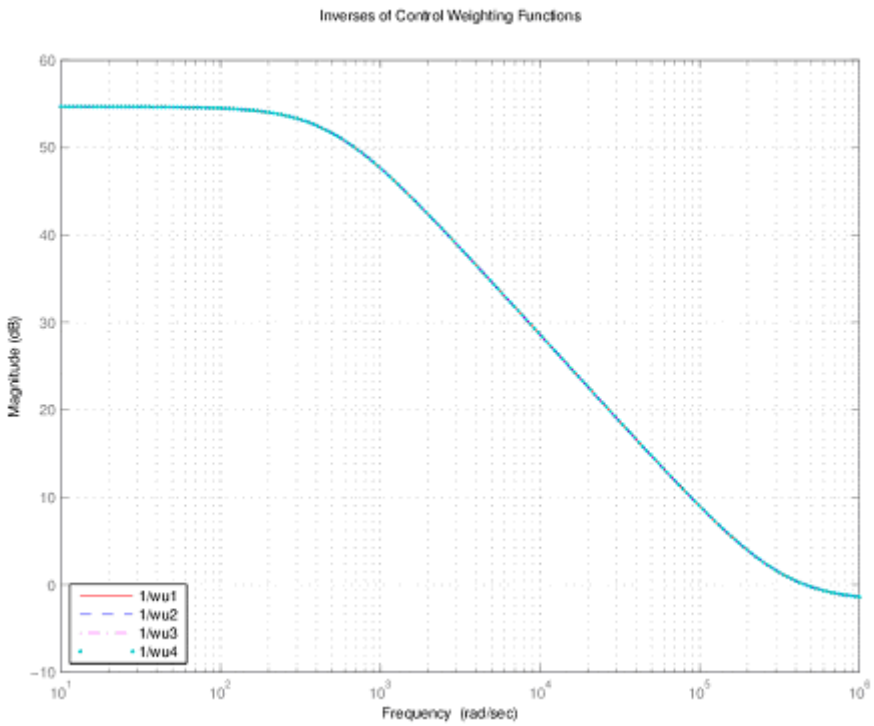



Figure 37: Inverse of Control Weighting Functions
Case 3

New lower values of the gain of control weighting functions move inverse control weighting function upwards, perhaps these changes will help to get the curve of control weighting function below of the curves that are representing the system with uncertainties. This will be checked later.

New values of μ show that results have improved to the point that now they are really near to the objective. Specifically in this last case the last value got after four iterations with μ-synthesis by D-K iterations is μ=1.078. Like in the cases before, the order of the controller is 22.

***Iteration Summary***

***----------***

| | | | | |
|---|---|---|---|---|
| ***Iteration #*** | ***1*** | ***2*** | ***3*** | ***4*** |
| ***Controller Order*** | ***22*** | ***22*** | ***22*** | ***22*** |
| ***Total D-Scale Order*** | ***0*** | ***0*** | ***0*** | ***0*** |
| ***Gamma Acheived*** | ***6400.00*** | ***25838.60*** | ***531.804*** | ***1.852*** |
| ***Peak mu-Value*** | ***98.942*** | ***2.766*** | ***1.211*** | ***1.078*** |

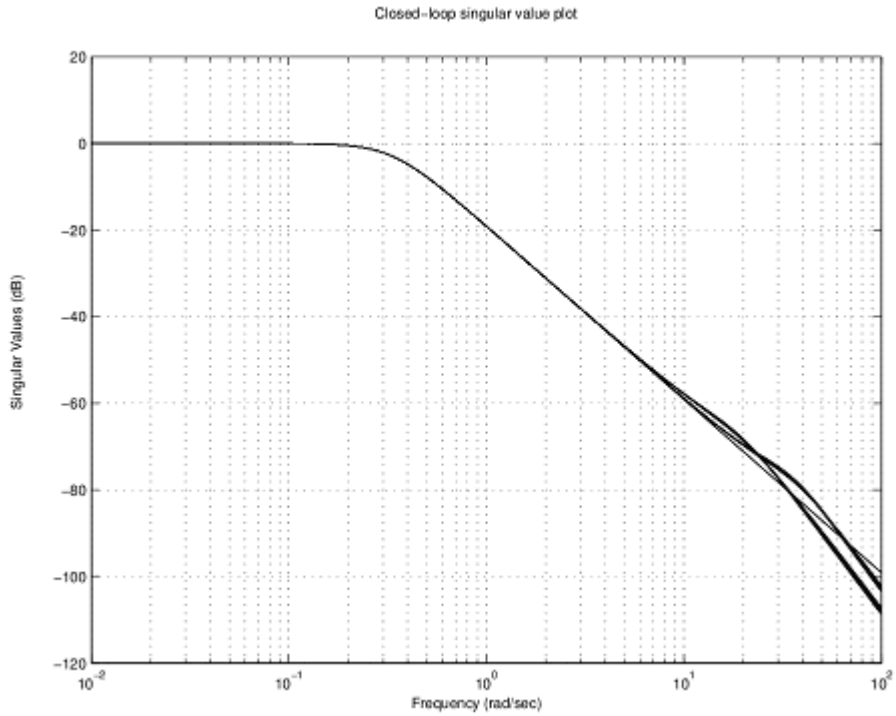



Figure 38: Singular value of the closed-loop system and desired response Case 3

In the third case still continue the difference between the desired behaviour of the system and the behaviour that is got. It seems that with these small changes in gains of the weighting functions desired behaviour does not change too much. As in the previous case, it is advisable to check input and output sensitivity functions in order to see easier if there is any improvement.

Taking a look in the figures below, it can be appreciated that in both cases there is improvement, namely, discontinuous curves that are representing weighting functions moved upwards, but in the case of the input sensitivity function, there is still a frequency range in which discontinuous curve is near to the continuous lines that are representing the system.

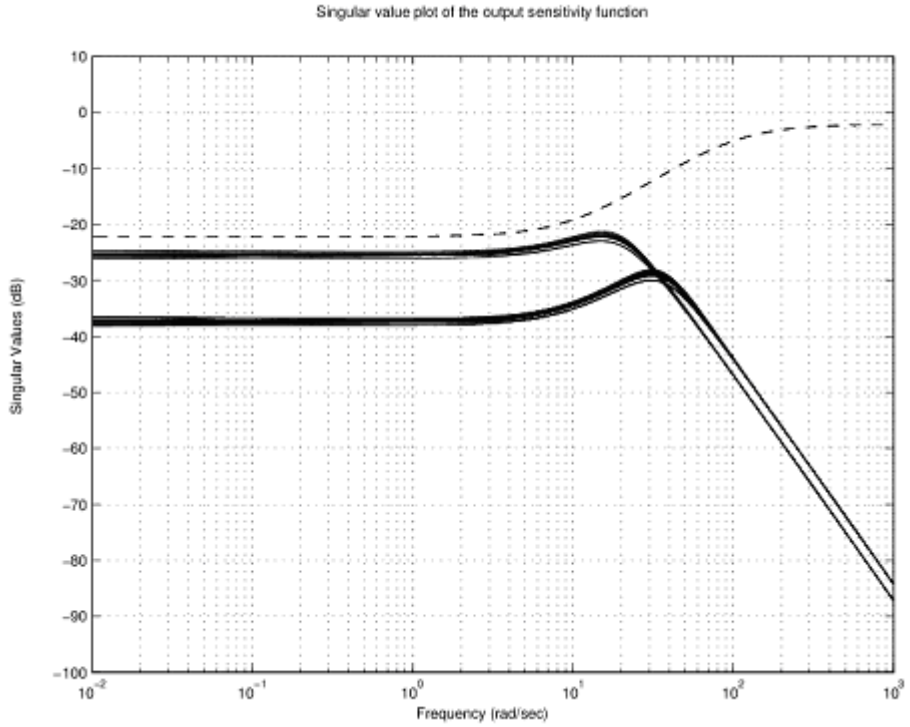



Figure 39: Output sensitivity function and inverse of Performance Weighting Function Case 3

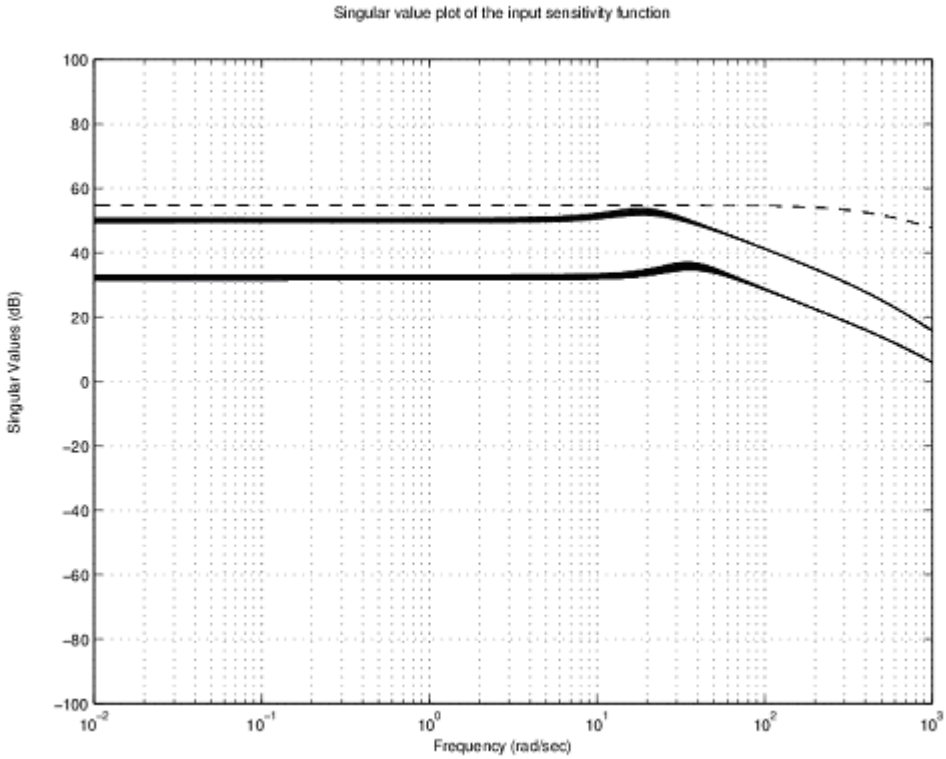



Figure 40: Input sensitivity function and inverse of Control Weighting Function Case 3

In order to get a value of μ lower than 1, another last change will be made in the last case.

### *3.3.4 Case 4*

For the last case the following gains are used for the weighting functions:

- $k_{pi} = 1.285$
- $k_{ui} = 0.00185$

As can be seen, the gain for the control weighting function is the same, so for this reason the plot of weighting functions will not be shown. In the case of the performance weighting function, gain is changed a bit in order to improve the value of μ, although, as it can be seen in the plot, the difference between the third and four case on inverse performance weighting function is not so big.

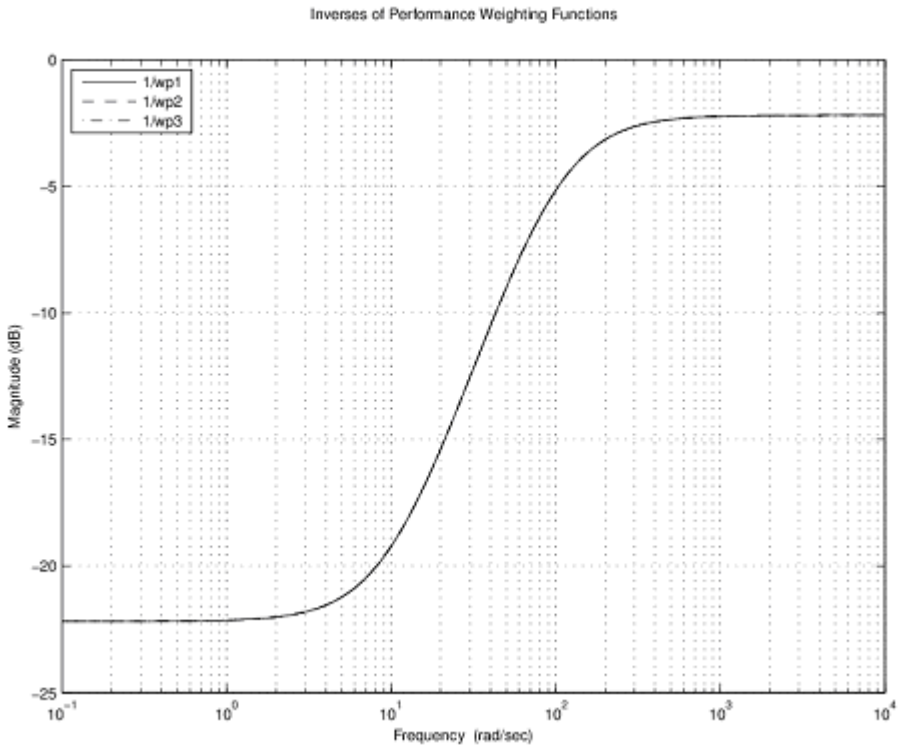



Figure 40: Inverse of Performance Weighting Functions Case 4

Although that the change are not so big, finally the value that is got after applying the μ-synthesis to this last case is 0.737. So now it can be said that is got a controller which fulfil the condition of robust performance and robust stability (μ<1). Results obtained in the application of the μ-synthesis to the last case are the following one:

***Iteration Summary***

```
-------------------------------------------------------
Iteration #            1       2       3       4
Controller Order       22      22      22      22
Total D-Scale Order    0       0       0       0
Gamma Acheived         6400.00 75.699  11578.80 0.743
Peak mu-Value          98.942  2.765   1.211   0.737
```

As can be appreciated in the data above, like in all the previous cases, the order of the controller is 22, so no complicated controller for computation.
Although that the order of the controller is the same as in the three previous cases, in this last case, as can be appreciated in the figure below, the controller that is got makes the system to have more or less the same behaviour than the desired system. In this case, no like the others, an improvement can be seen. Both are the same except a short part in the end.

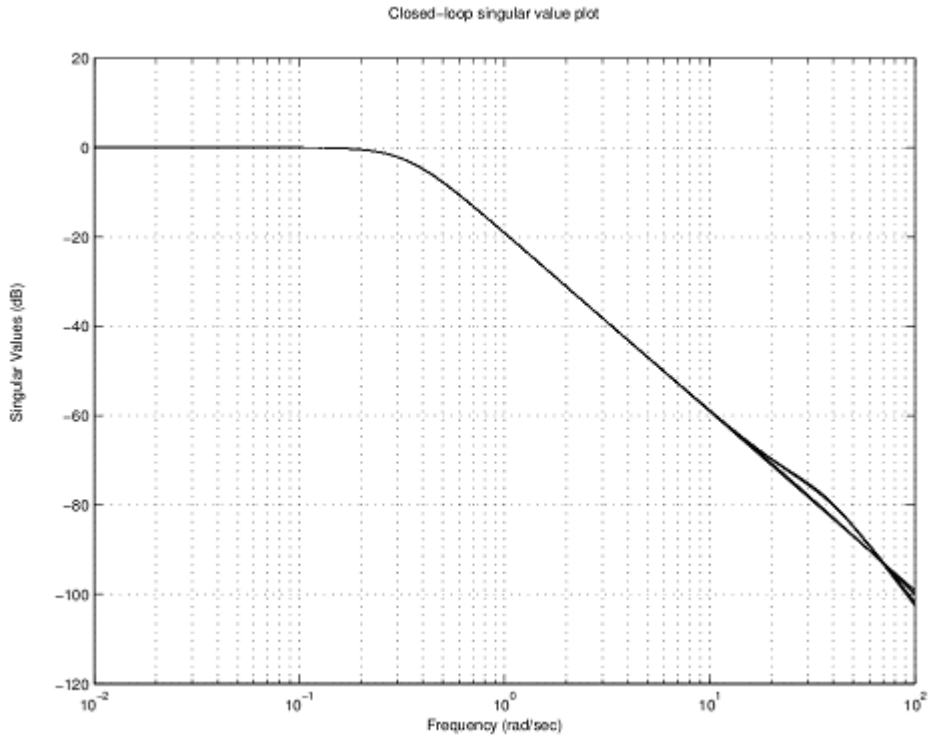



Figure 41: Singular value of the closed-loop system and desired response
Case 4

The improvement explained with this last plot it can be seen also in input and output sensitivity responses: where in both cases inverse of weighting functions are above the continuous lines that are representing the system with his uncertainties. Specially, it should be noted the big difference that is got in the plot of the output sensitivity function, where the discontinuous line representing the inverse of performance weighting function is higher than on the rest of the cases. This is due to the reason that the procedures are numerically sensitive to small changes because of the conditioning of the matrices.

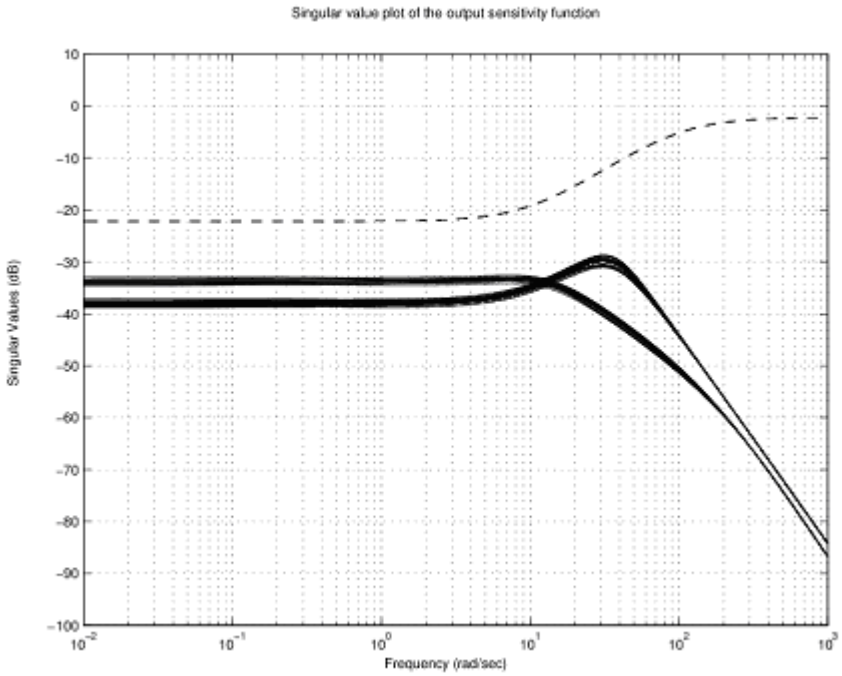



Figure 42: Output sensitivity function and inverse of Performance Weighting Function Case 4

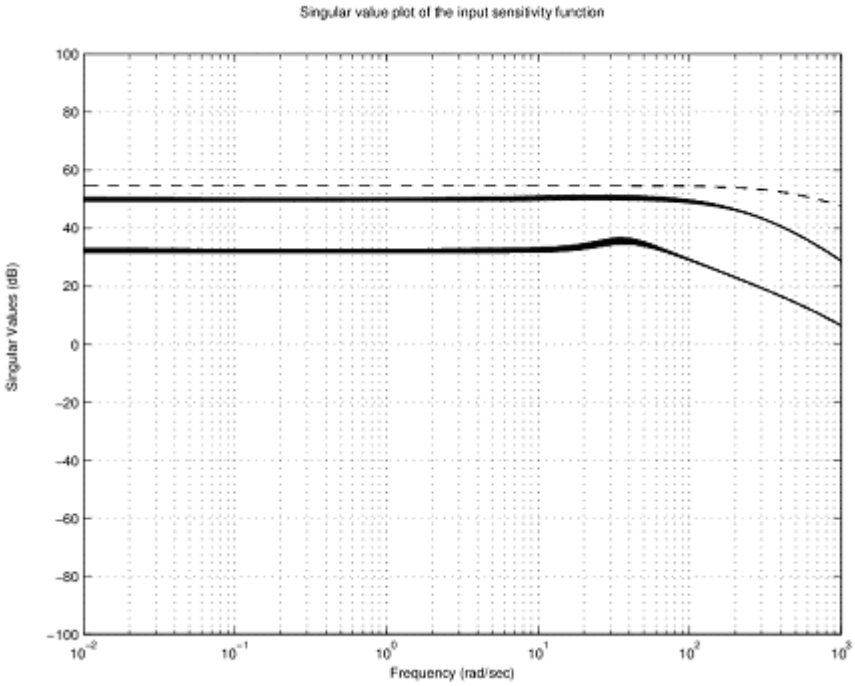



Figure 43: Input sensitivity function and inverse of Control Weighting Function Case 4

# Chapter 4: Simulation results

## 4.1 Comparison of cases

In the following chapter different results of simulation are shown.
At first, certain position is established and a simulation is done for two different case of controller, the first and the four case of the previous chapter exactly.
After wind disturbances are introduced in the system in order to see how the system reacts to those signals. Finally, to this last system, some signals are introduced in order to simulate measurement noise in the feedback.

In the table below are represented departure point and desired position during the hovering of the quad-rotor:

| | Departure point | Desired point |
|---|---|---|
| X | 0 | 200 |
| Y | 0 | 10 |
| Z | 0 | -5 |

Table 4: departure position and desired arrival position

The main is to get the desired position as fast as possible taking account that the behaviour of the system has to be stable during all the flight of the quad-rotor.

### 4.1.1 Case 1

If the simulation of the Simulink is done with the desired position above with the first controller, the following results are got:

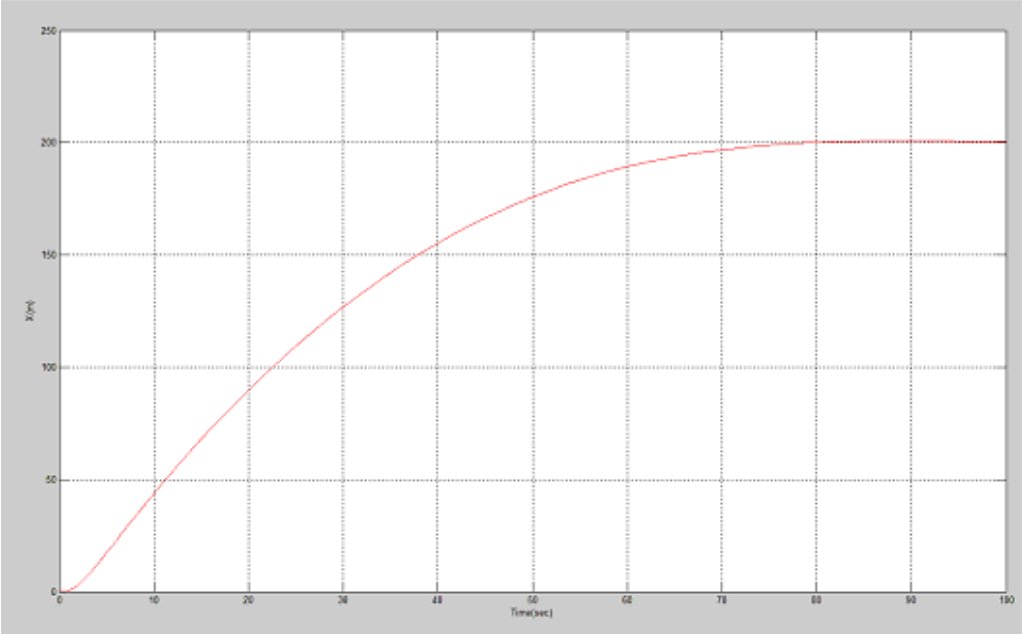



Figure 44: Response in x
Case 1

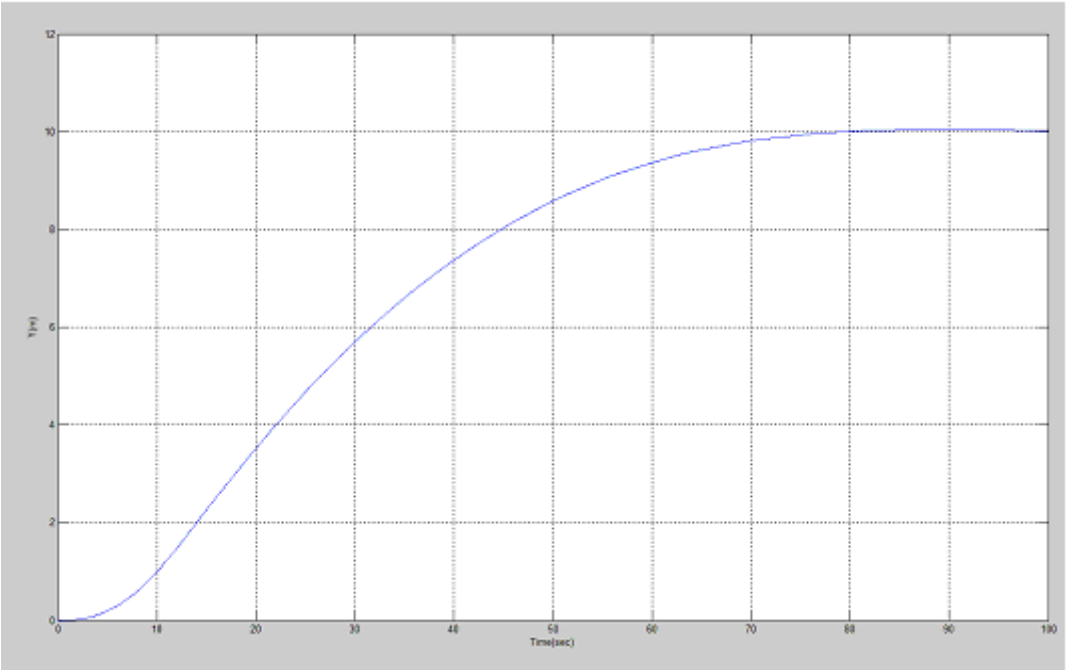



Figure 45: Response in y
Case 1

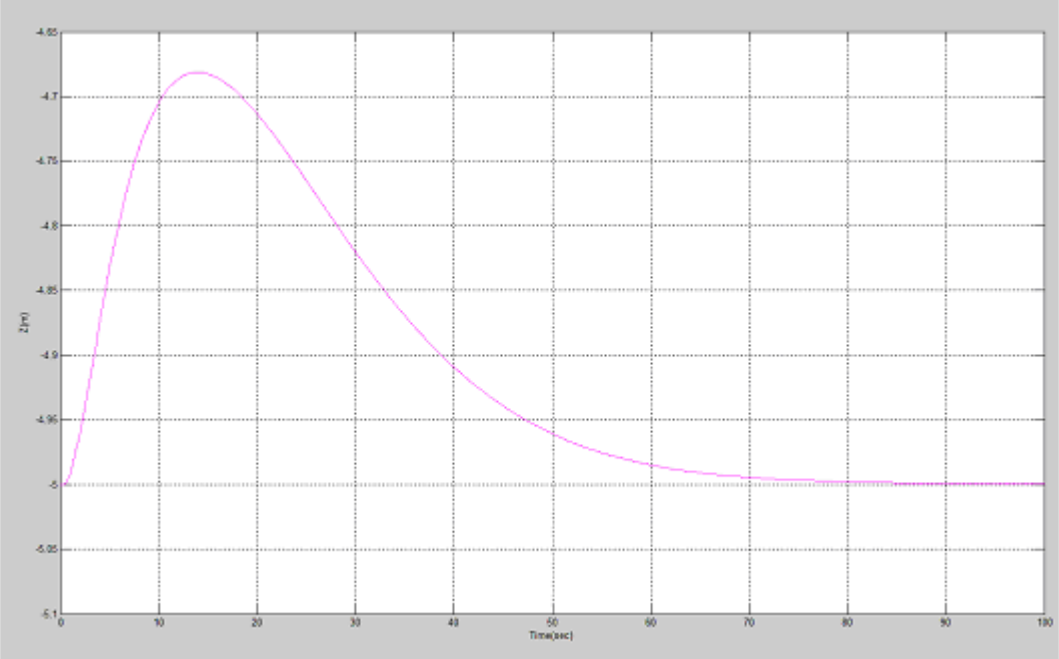



Figure 46: Response in z
Case 1

As can be seen in plot of the coordinates, the desired position is got when time arrives to eighty seconds. It has to be taken in a count that in the case of the z coordinate is negative due to the fact that the positive axis is in the direction of the gravity.

In the case of the angles, the controller gets the objective of keeping them next to trim values: only at the beginning some values are remarkable, above all the case of the pitch angle where it arrives to value higher that 0.25 radian.

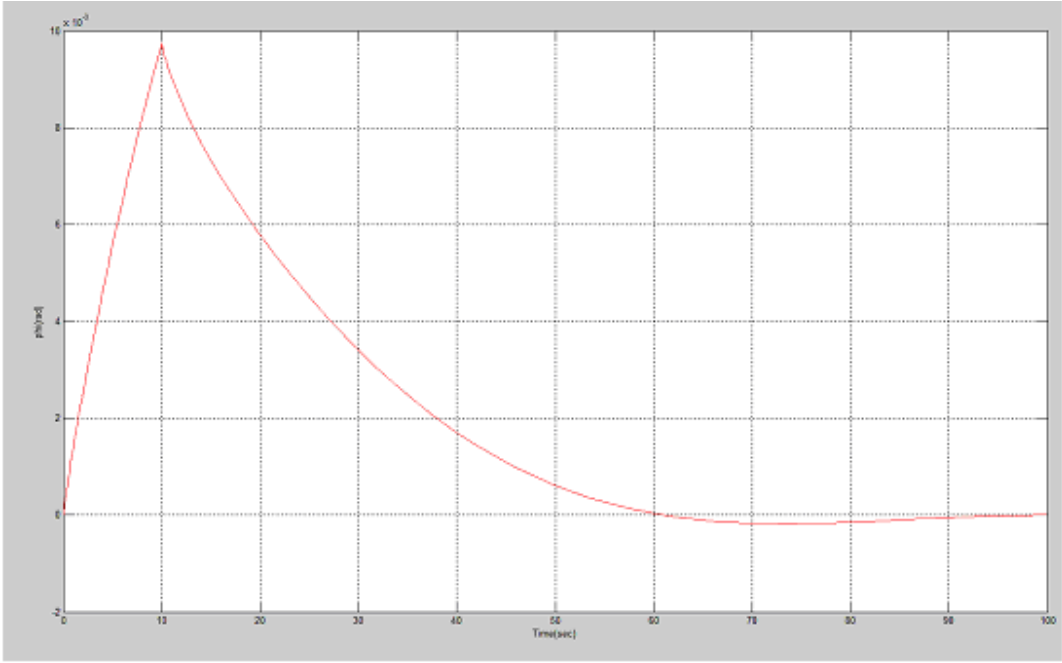


Figure 47: Response in φ
Case 1

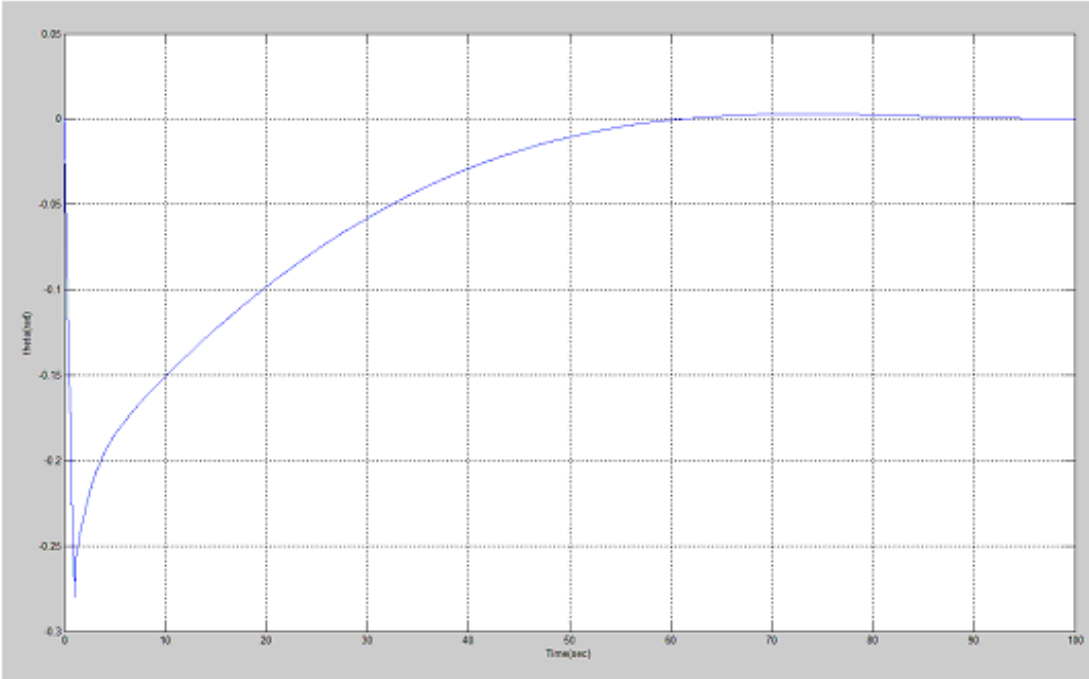


Figure 48: Response in θ
Case 1

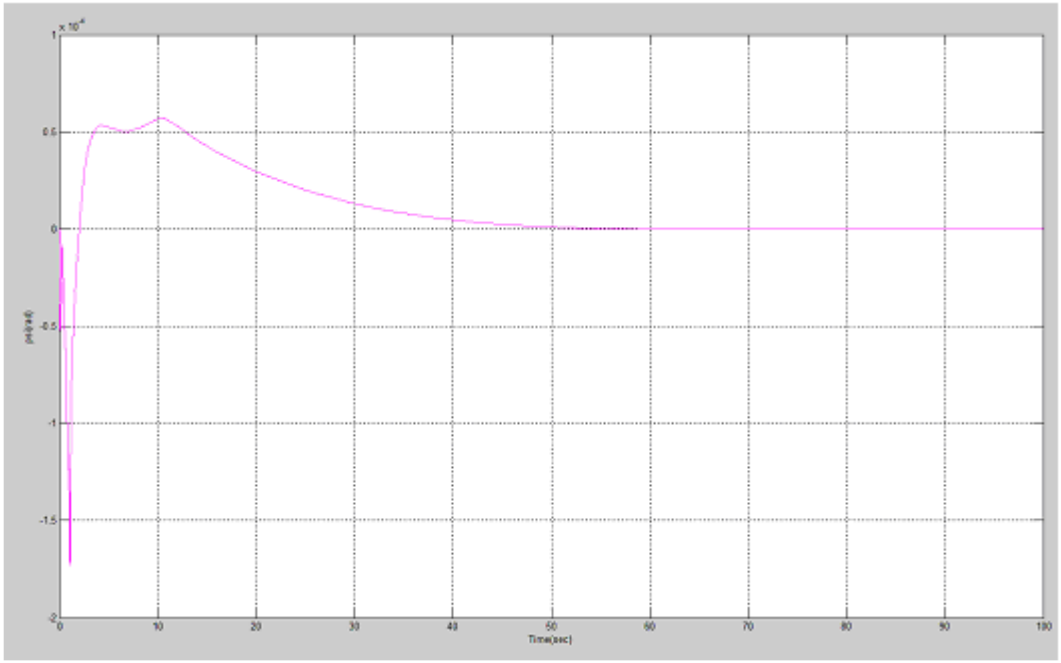


Figure 49: Response in ψ
Case 1

### *4.1.2 Case 4*

If values of the weighting functions are changed to the gains used in the fourth case, these are the following results that are got in the simulation:

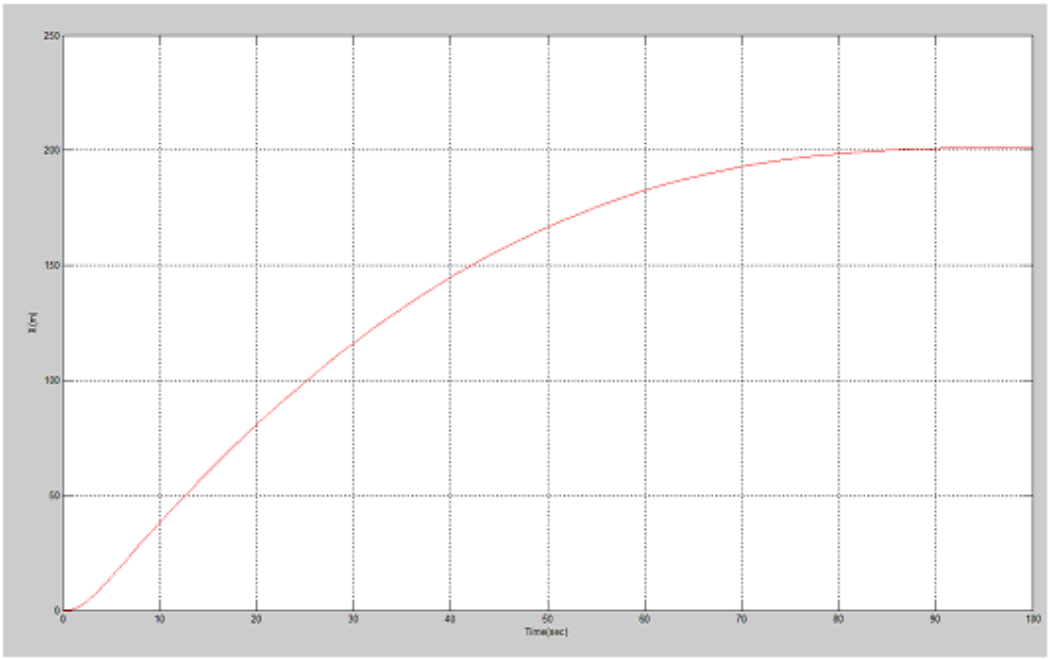


Figure 50: Response in x
Case 4

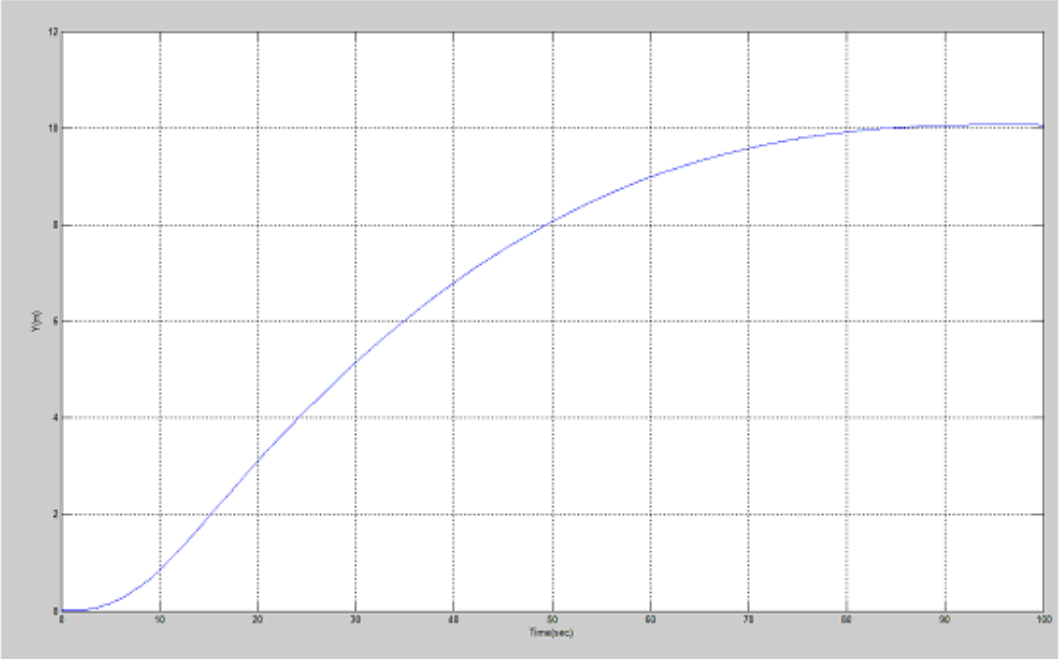



Figure 51: Response in y
Case 4

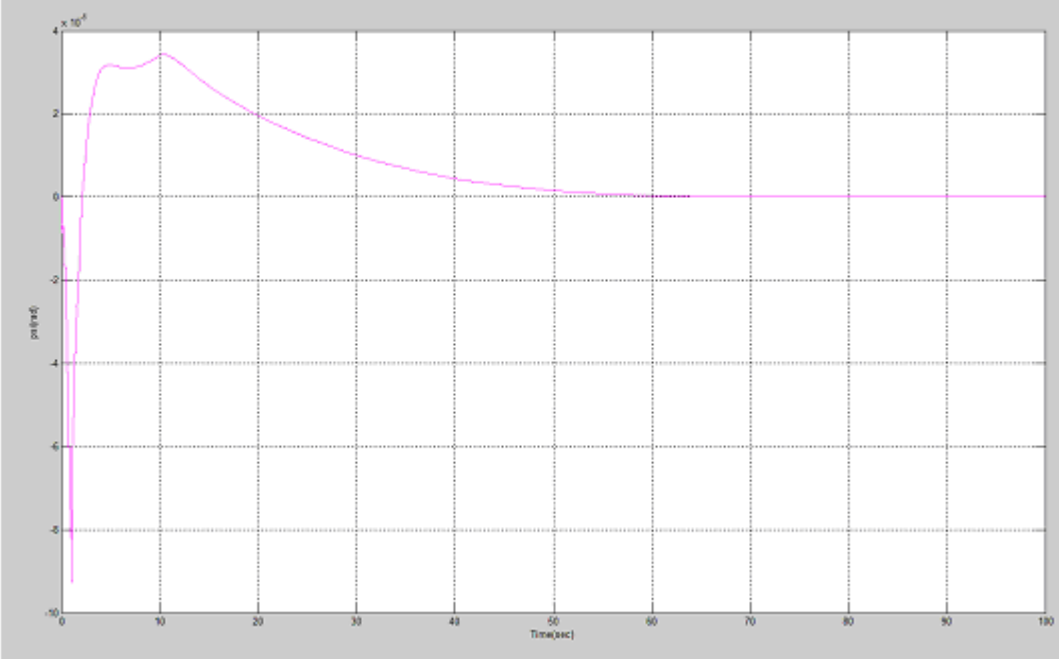



Figure 52: Response in z
Case 4

As in the first case, in this one also the desire position is got, but there are some facts that should be remarked.
In the one hand, comparing the x plot of both cases, it is possible to see that the first one gets the desired position in less time than the second one, due to the fact that the dynamics of the first controller are faster than the dynamics of the fourth controller.

In the other hand, for the case of the z coordinate, in the second case the maximum deviation respect to the trim value during the simulation is smaller than the first case. This fact happens due to the reason that the controller of the four cases is better than the controller of the first one as it is explained in the previous chapter.

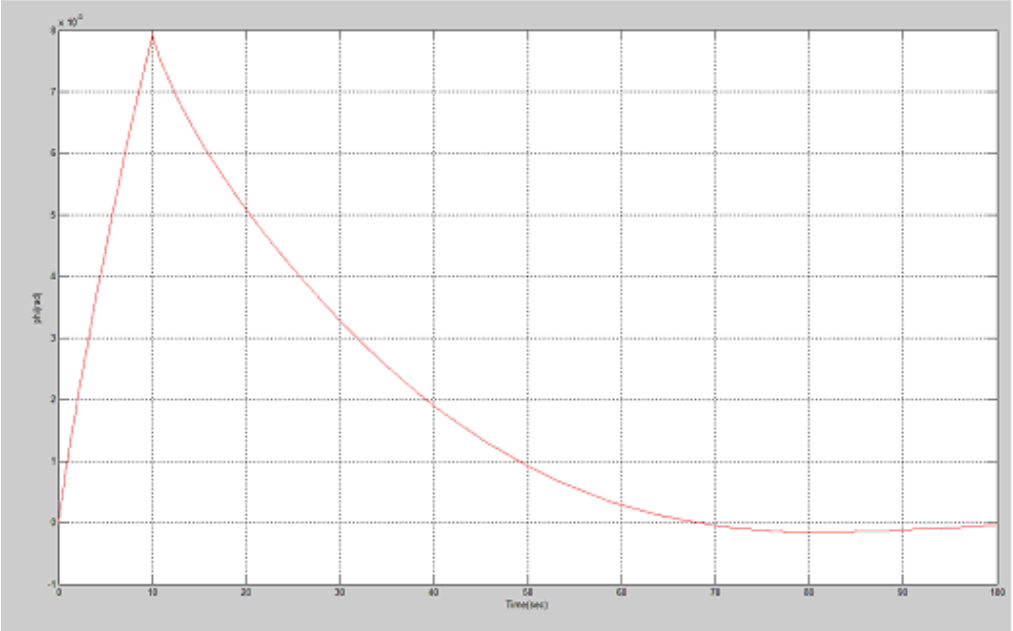


Figure 53: Response in φ
Case 4

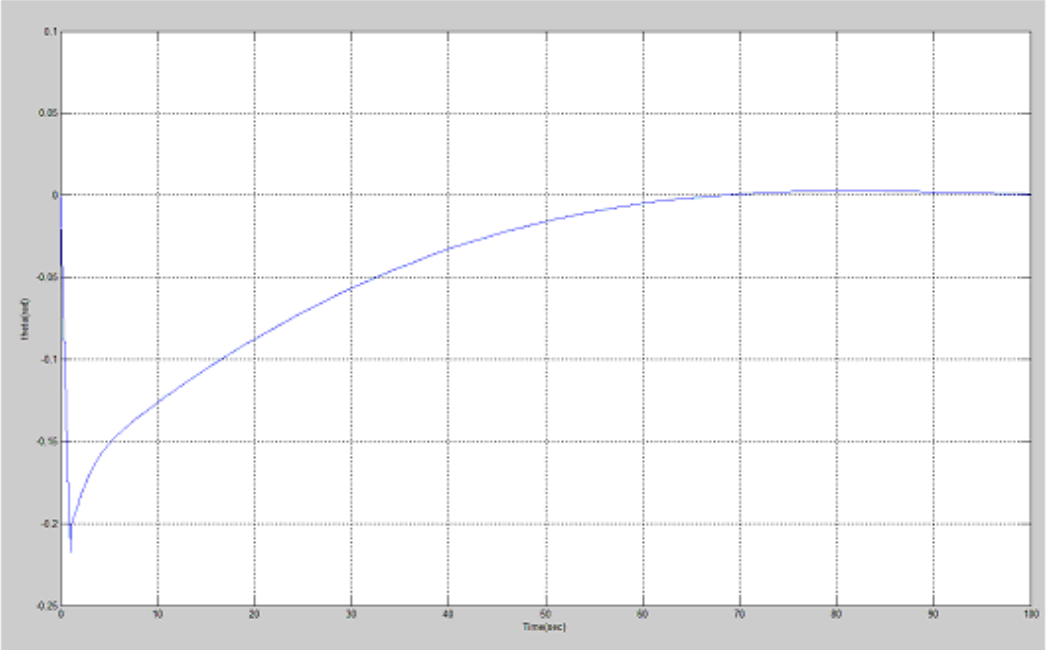


Figure 54: Response in θ
Case 4

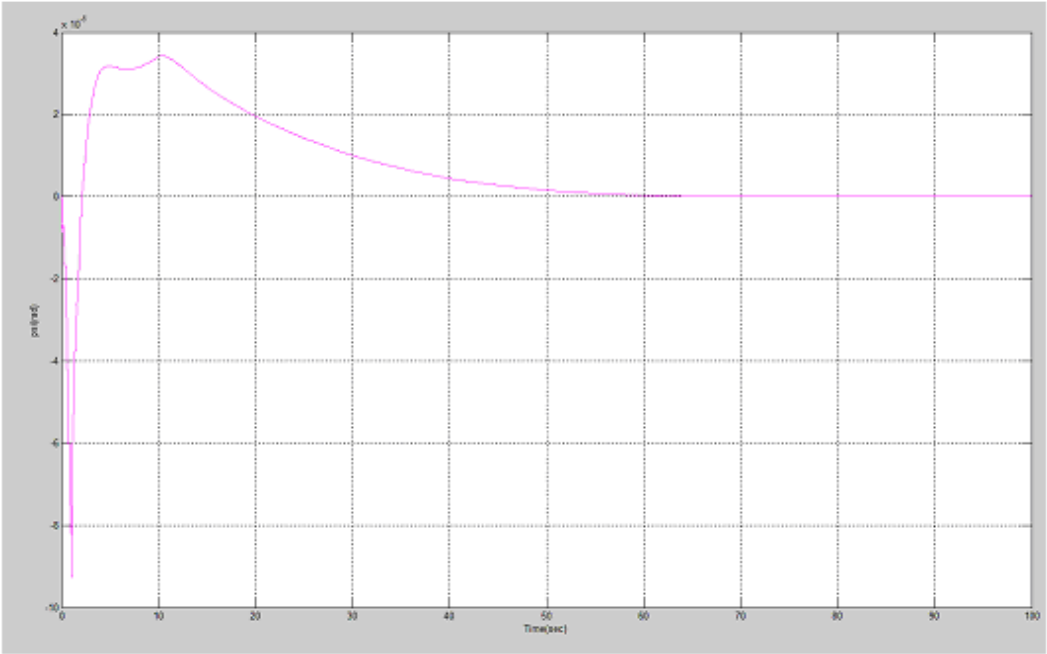


Figure 55: Response in ψ
Case 4

In the case of the roll, pitch and yaw angle something similar to the case of the coordinate z happens. In three cases maximum deviation is smaller in the second case than in the first case due to the robustness of the controller. Moreover, like in the case of the x coordinate, in plots of the angles it can be seen that for the first case the responses are faster that in the second case: in other words, the desired position is got in a shorter time.

## *4.2 System with wind disturbances:*

After testing that the model is working properly, the next step is to check how this model reacts to the introduction of wind disturbances in different directions of the body frame, in order to make the system more real. For simulating wind in different directions, three signal generators are used, one per each axis, as can be seen in the Simulink model. All of them are different, with different amplitude and frequency due to the fact that it is wanted to simulate the randomness of the wind. These are the signals used in this paper:

| | U_wind | V_wind | W_wind |
|---|---|---|---|
| Signal type | Square | Square | Square |
| Amplitude | 0.2 | 0.18 | 0.04 |
| Frequency | 0.05 | 0.04 | 0.03 |

Table 5: Signals used for the wind disturbances

It is explained in the first chapter that the disturbances normally appear in low frequency, this is the reason why the frequency chosen for the simulation of wind have small values.
The following results are got after the simulation:

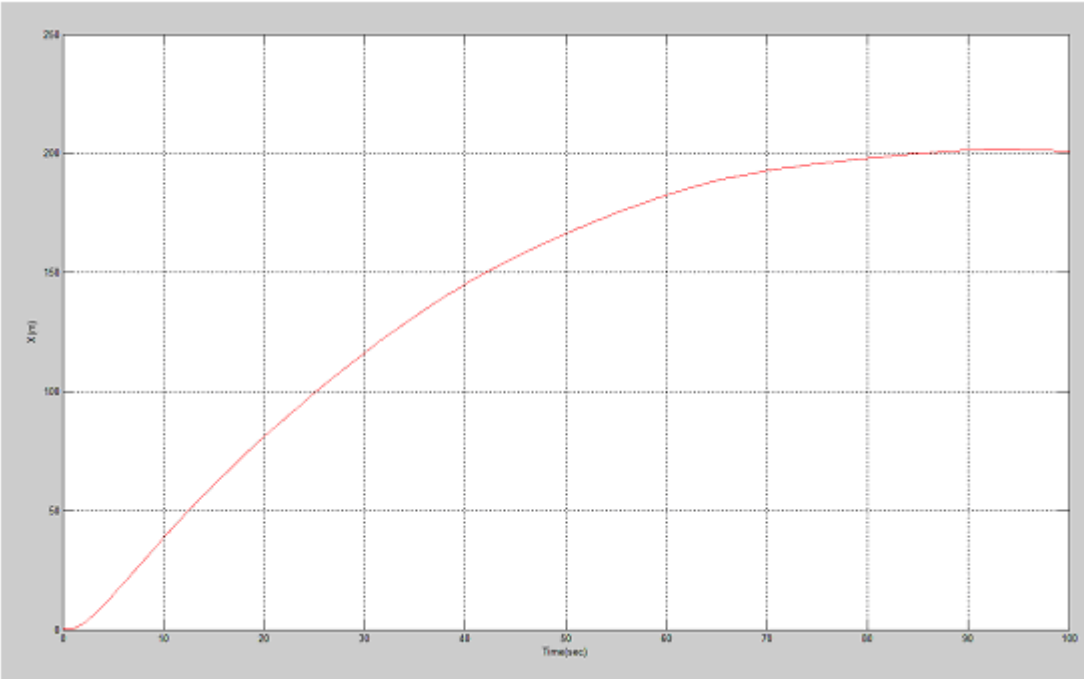



Figure 56: Response in x with wind disturbances

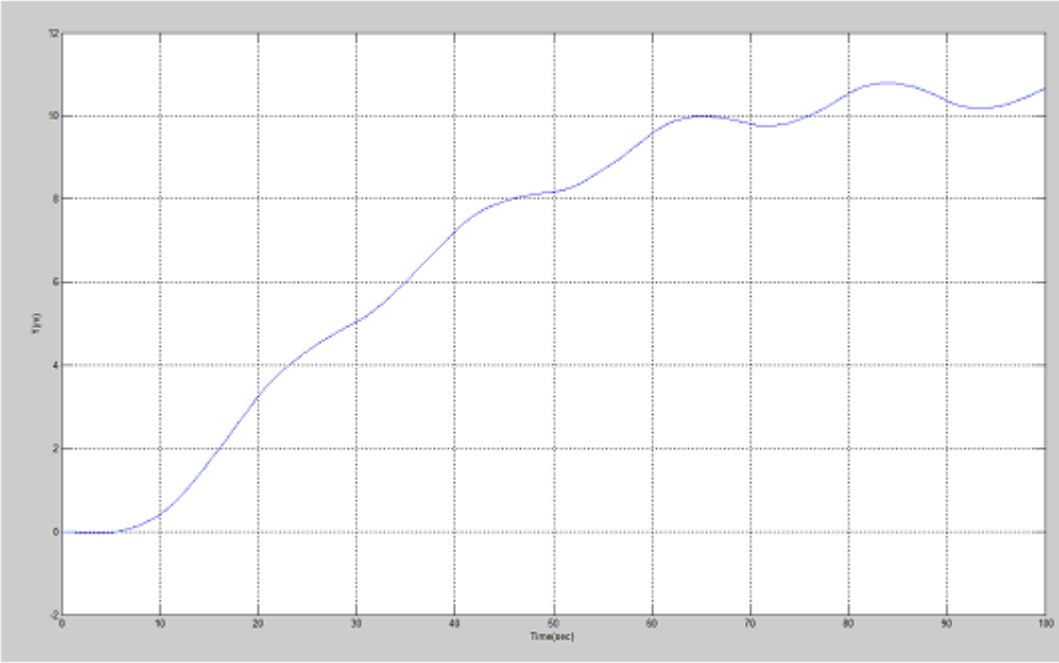



Figure 57: Response in y with wind disturbances

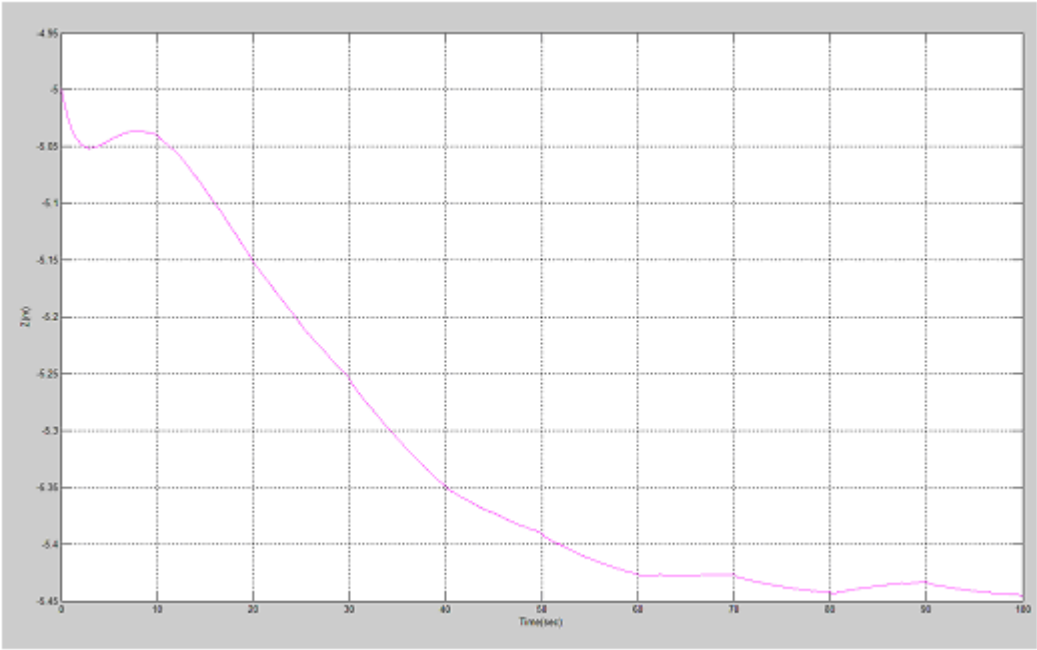


Figure 58: Response in z with wind disturbances

As can be seen in plots above, some oscillations appear during the flight of the device due to the signals that are introduced for simulating the disturbances of the wind; although, desired position is got, with certain deviations reference value like in the cases of the *y* and *z*.

Something similar happens in the case of the angles. Here is where the controller got applying μ-synthesis has to work harder because of the disturbances of the wind that are changing all the time roll, pitch and yaw angles. These disturbances are all the time changing these values that the controller has to keep near to trim values.

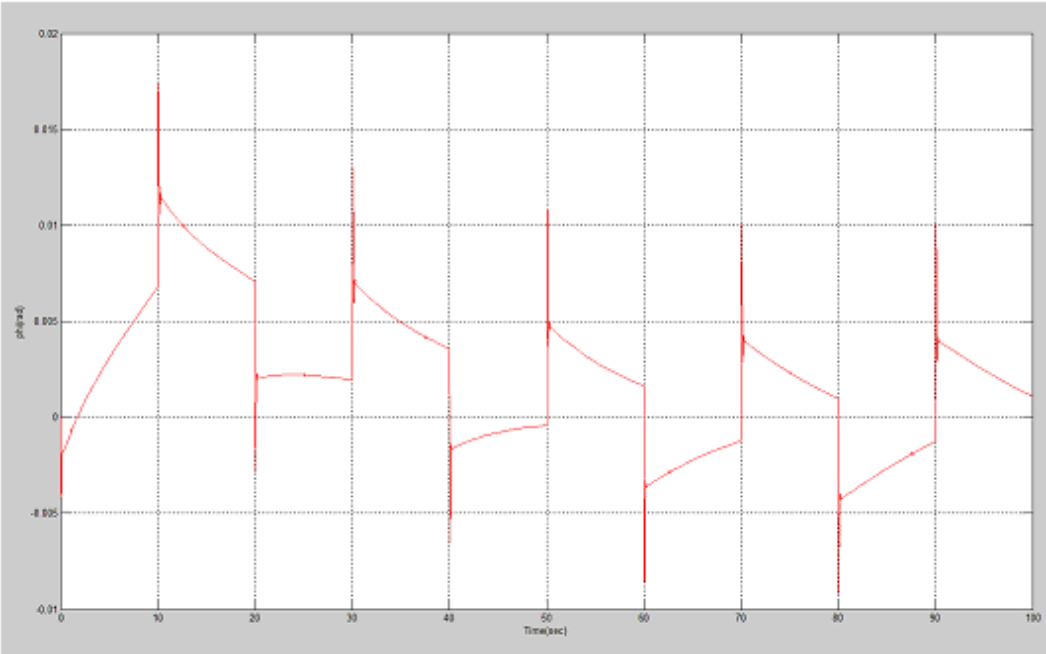


Figure 59: Response in φ with wind disturbances

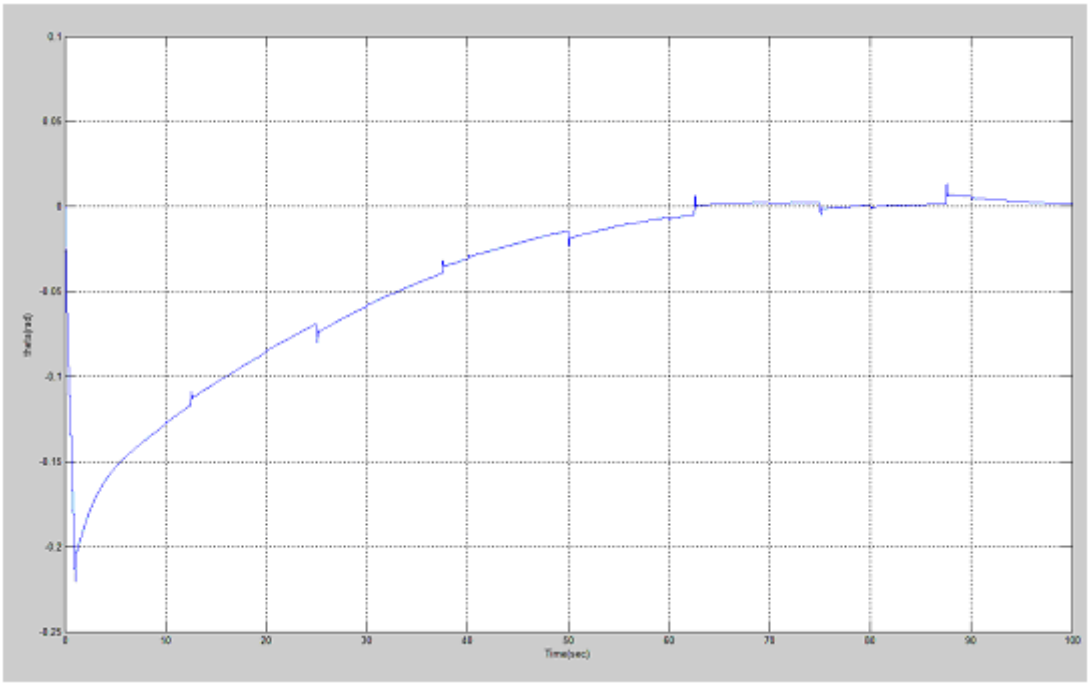


Figure 60: Response in θ with wind disturbances

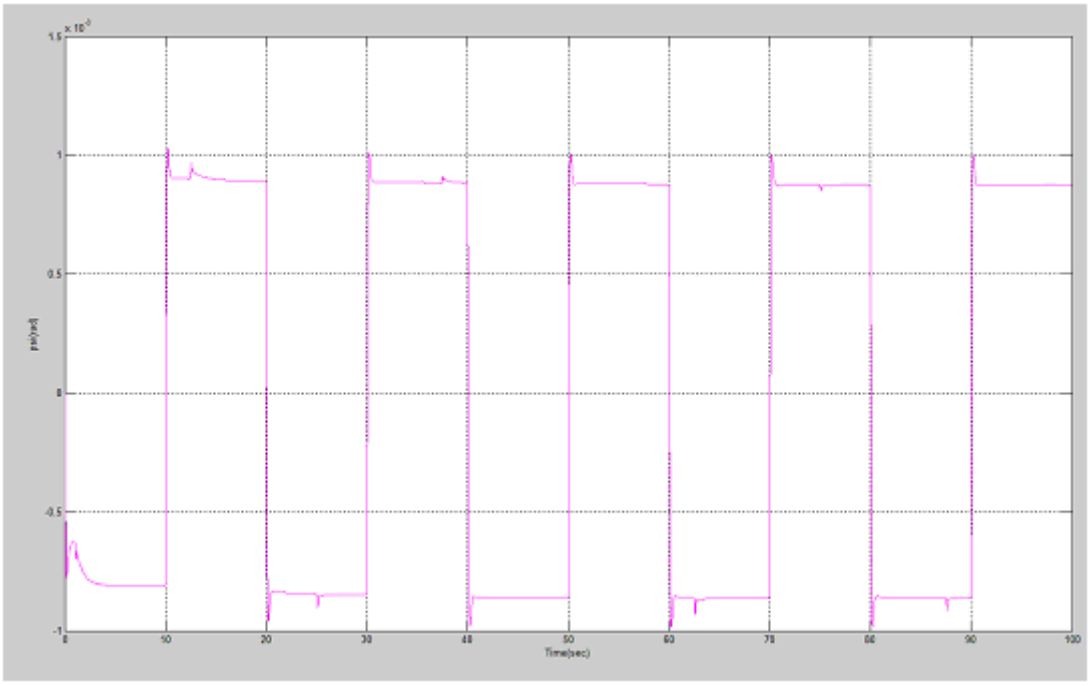


Figure 61: Response in ψ with wind disturbances

In previous plots is possible to check how deviations of the values of the angles are not so high in reference to the trim values that are wanted. Is remarkable one of the values that is got in the figure 60, where *theta* has value a value bigger than -0.2rad. Apart of that, other fact that is remarkable is the periodicity of the yaw angle due to the signal that it is used.

## *4.3 Influence of the measurement noise on feedback*

Once that is checked the behaviour of the device with the influence of the wind, the next and last step is taking account the noise that appear in high frequency due to the feedback that is made.

In this case, taking the Simulink showed in the second chapter, on the feedback of angular body rates noise is introduced using three band-limited band noise generators, once per each body rate. For choosing the power of the noise that is going to be used, it is necessary to check which are the magnitudes that are measured in the roll, pitch and yaw rates, in order to see which it should be the magnitude of the noise in the generators. Normally the noise is not bigger than 10-20% of the magnitude that is measuring.

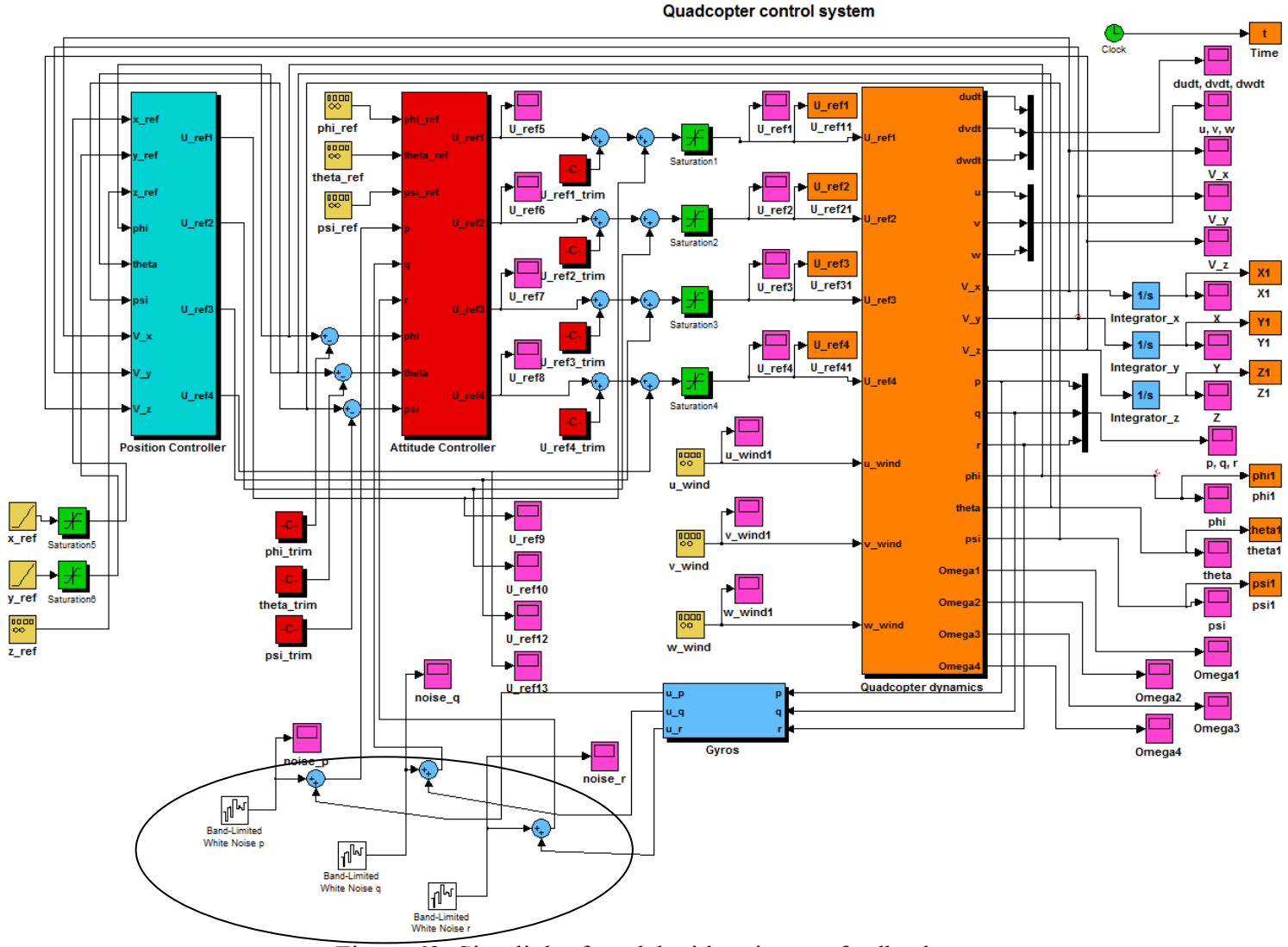



Figure 62: Simulink of model with noises on feedback

These are plot got after the new simulation with noise:

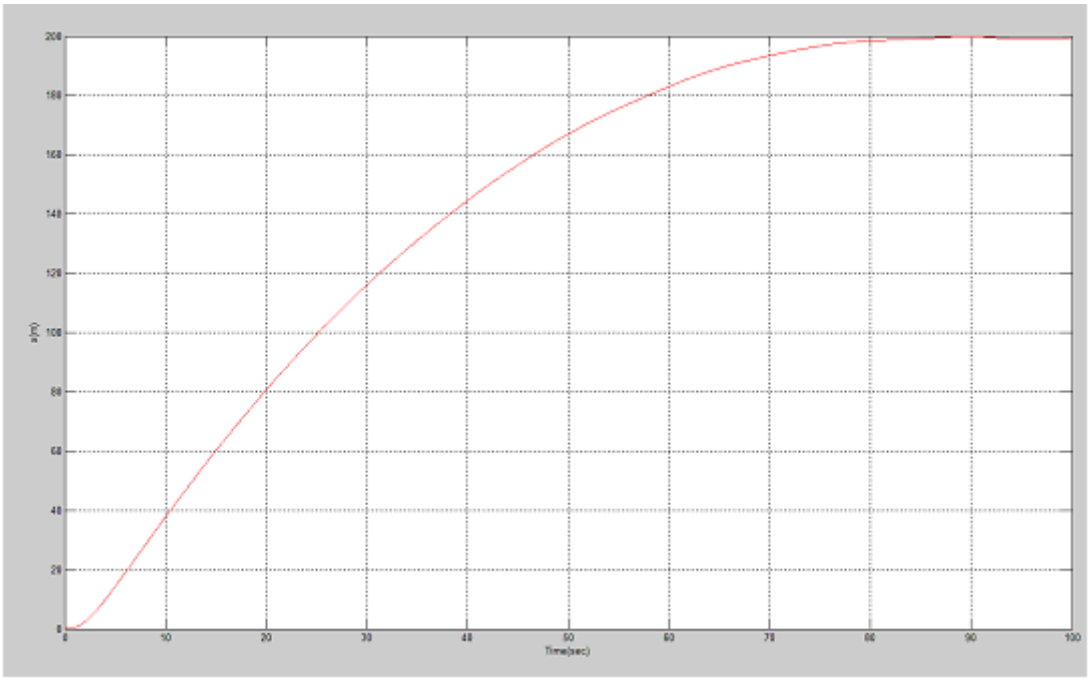



Figure 63: Response in x with wind disturbances and noise

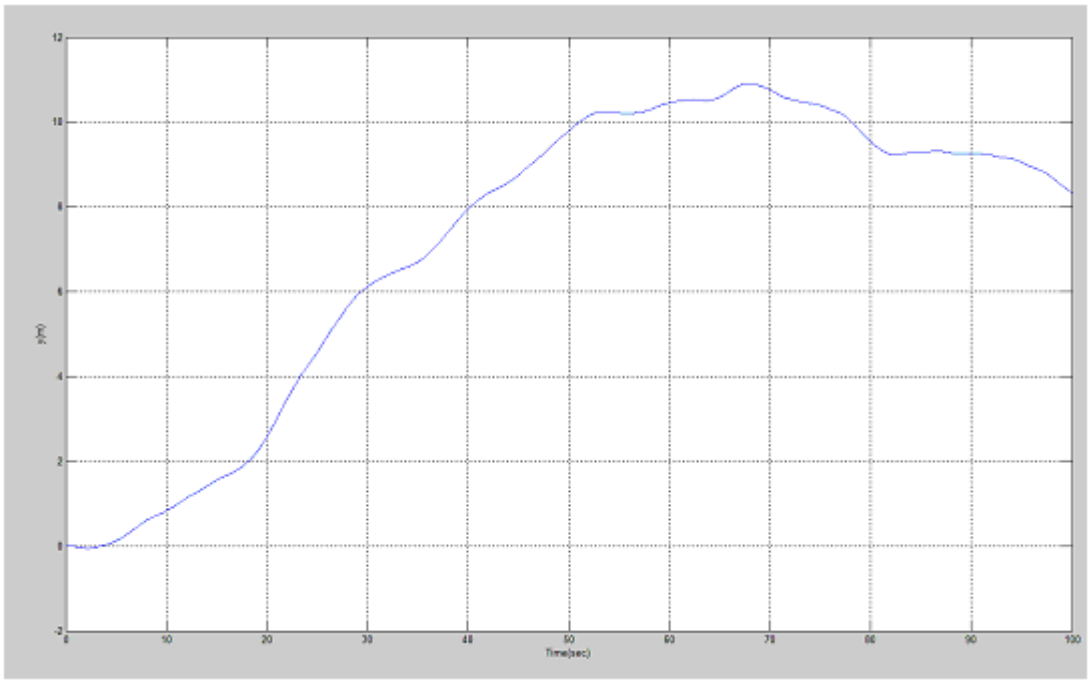



Figure 64: Response in y with wind disturbances and noise

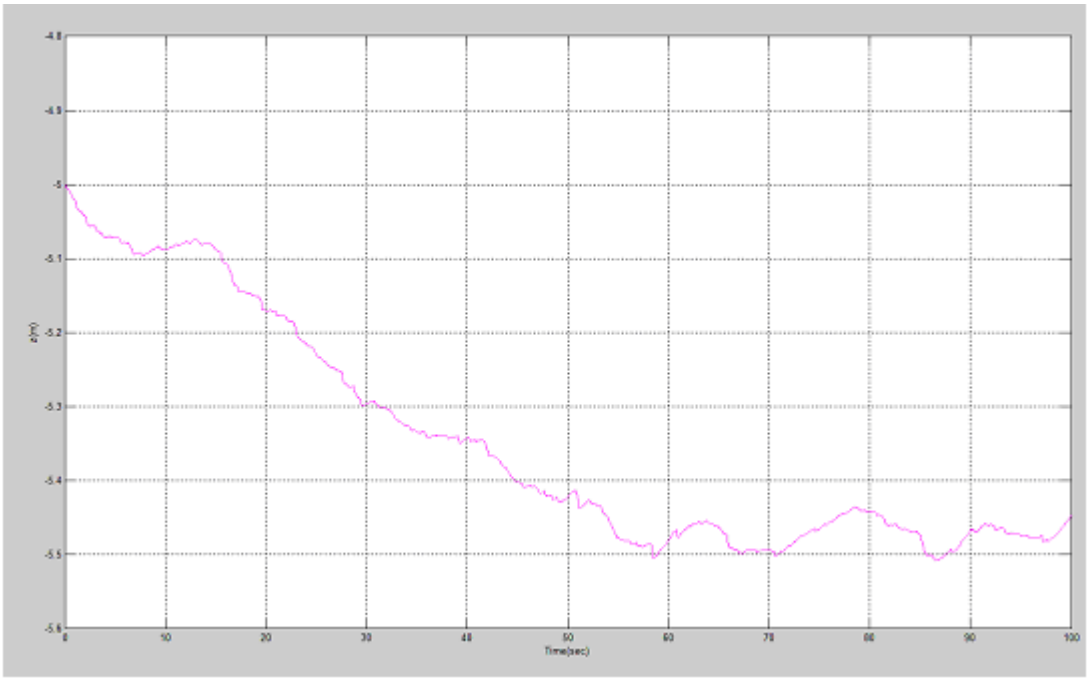



Figure 65: Response in z with wind disturbances and noise

Plots above show that despite of the noise that is introduced in the measurement, the desired values for the position that are wanted for three coordinates are got. It can be see that in the case of the x coordinate the difference with the case of the wind disturbance is not so big, but in the cases of y and z the difference is bigger. It looks like for the case of y the objective is got in a faster way than in the previous case, but at the same time the deviations are bigger at the same time that they are acceptable.

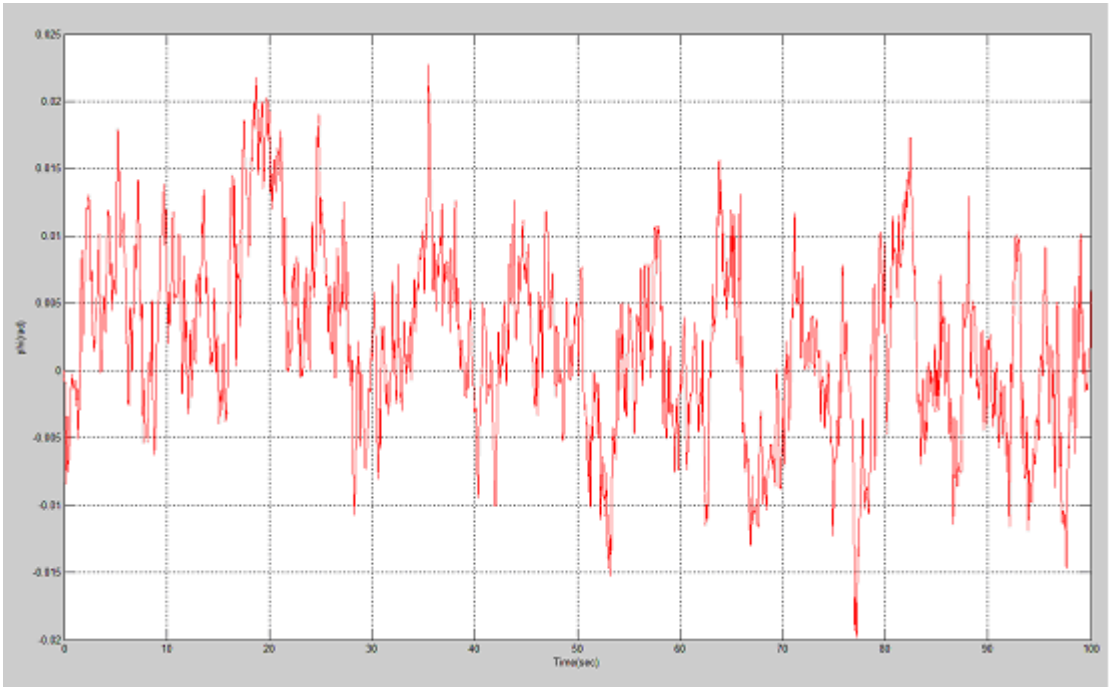



Figure 66: Response in φ with wind disturbances and noise

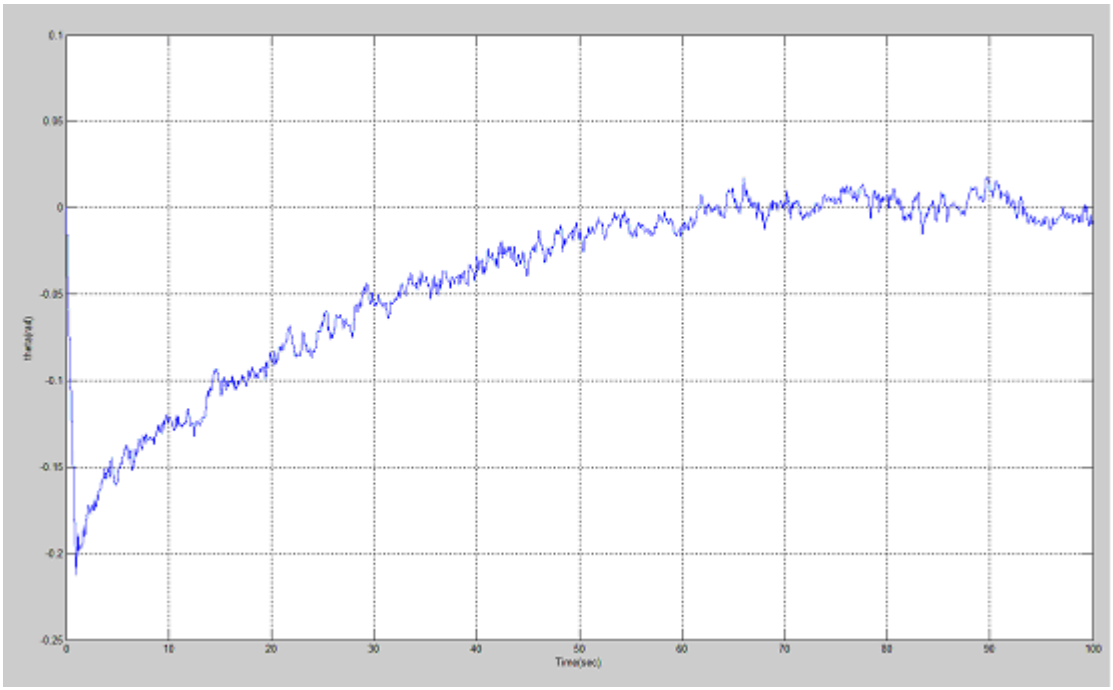



Figure 67: Response in θ with wind disturbances and noise

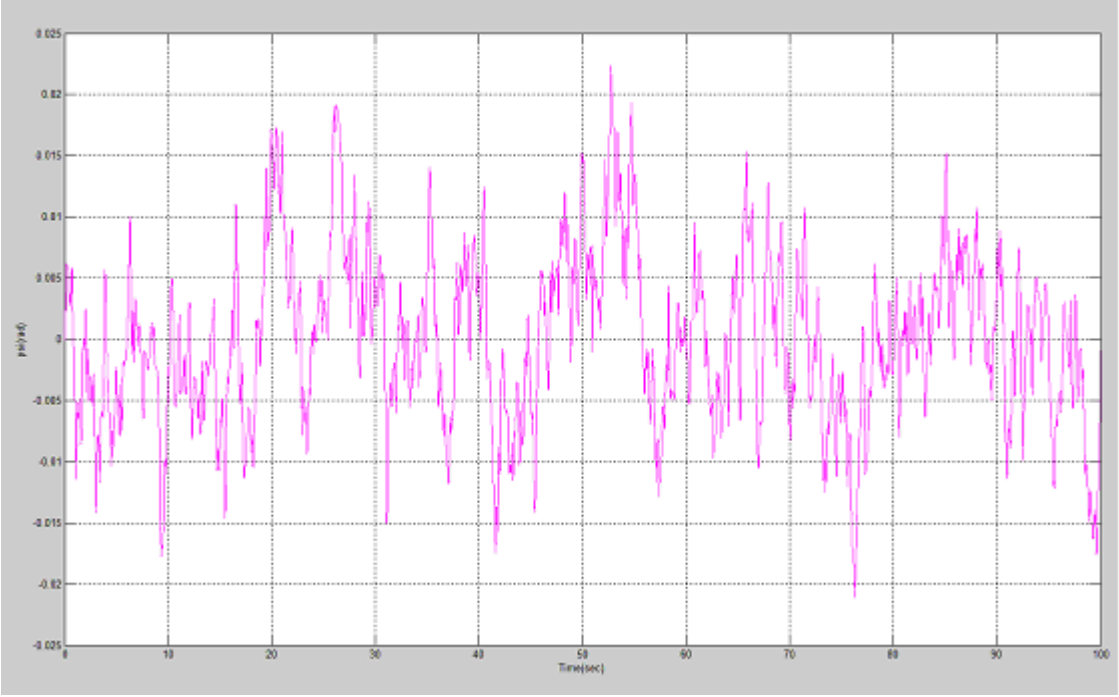



Figure 68: Response in ψ with wind disturbances and noise

It is in the case of the roll, pitch and yaw angles where it can be appreciated further the influence of the noise. As can be seen in plots above, despite of the noise, trim values are required for the good operation of the device. In contradistinction to the case of the wind, in the case of roll and yaw angles, deviation are bigger due to the influence of the noise.

### *4.4 Comparison of both cases*:

In order to see the difference in the following plots both cases are represented together.

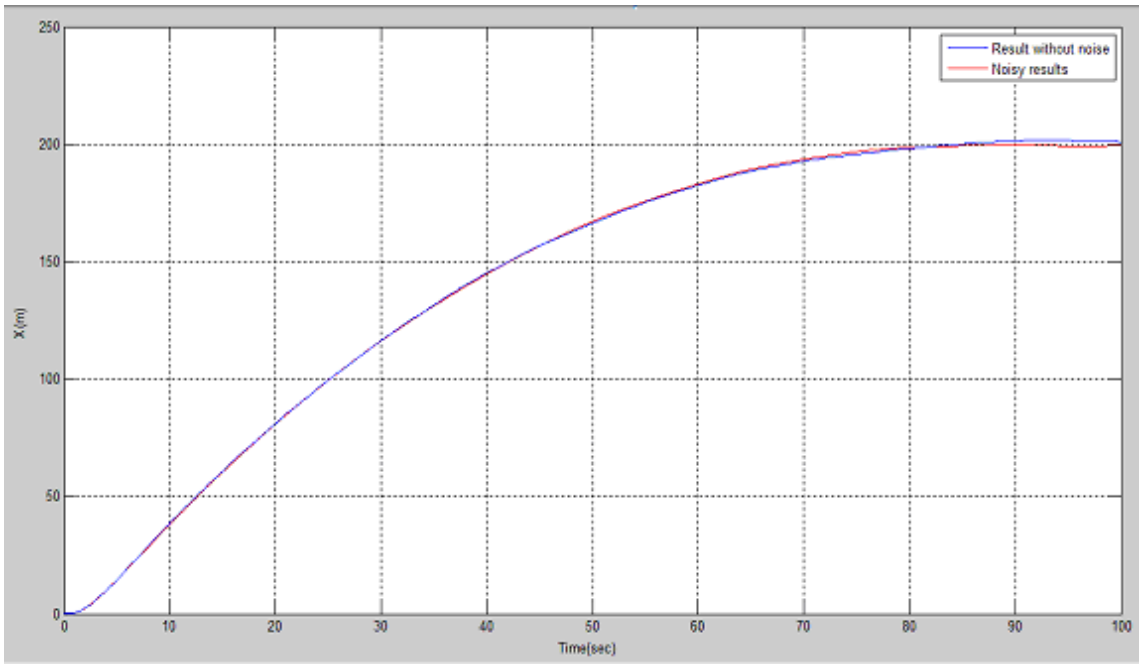



Figure 69: Response in x for both cases

In the plot above of the x coordinate for two cases it is possible to see that the frequency responses in both cases is similar, and in the same time, similar to the case in which the disturbances and noises are not considered.

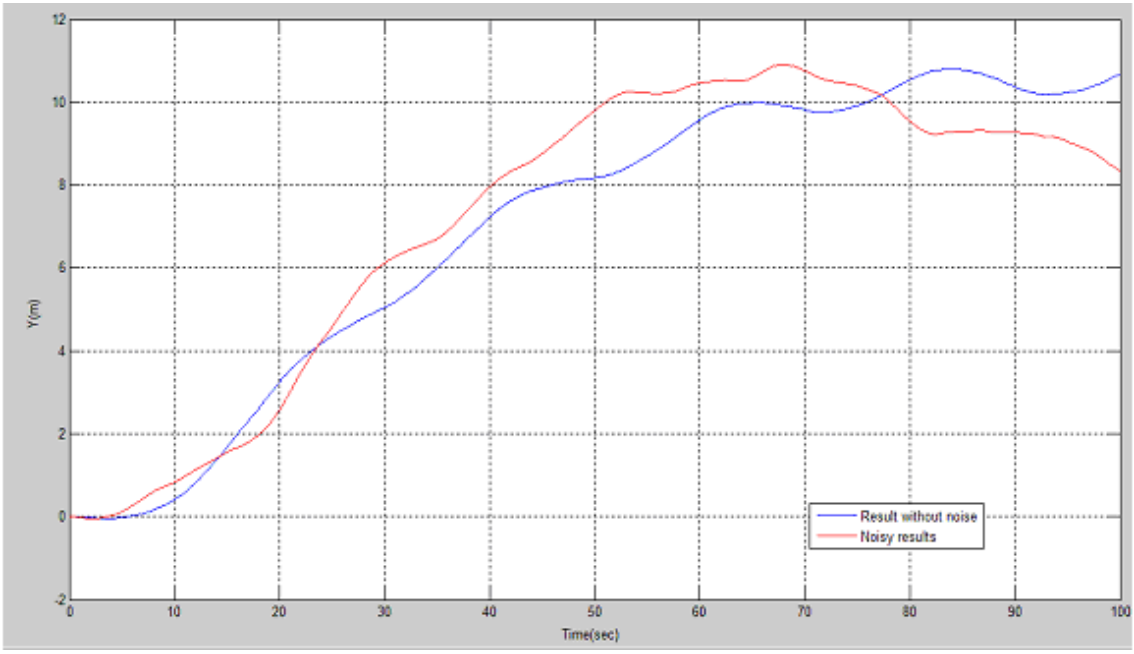



Figure 70: Response in y for both cases

In the case of the y coordinate, something similar to the case of the x coordinate happens, trim values are got but in the last case where the measurement noise is considered in a faster way, although, as it is expected, in the last part deviation is higher.

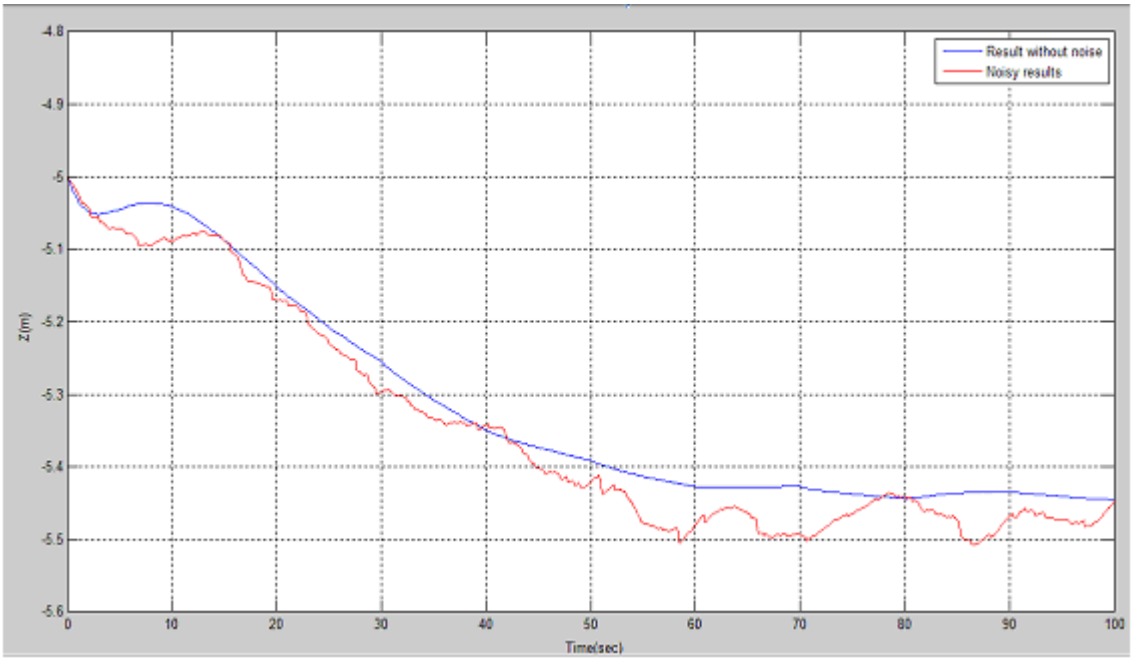



Figure 71: Response in z for both cases

For the z coordinate, the frequency response is similar in both cases, thus, wanted high is got, but should be noted that due to the noise in the last case the deviation is bigger although acceptable.

As can be seen in the plots above, difference between the both cases are not so big, something that is not happening in the cases of the angles, where the difference between the both cases are really different. This is due to the reason that when it is wanted to calculate the x, y and z coordinates, two integrators are introduced, one in the controller in order to calculate the velocity from the acceleration, and the other to calculate the position from the velocity calculated before. These integrators are making as if they were filters and these is the reason why the signals are not changing in the same way that they are changing in the case of the measurement noise.

Despite that there is not this kind of effect in the case of the rates, in both cases values of the angles are near to the trim values. For the case of wind disturbances deviation are smaller than for the case of the wind disturbance and noise together, but still al the values of deviations are acceptable.

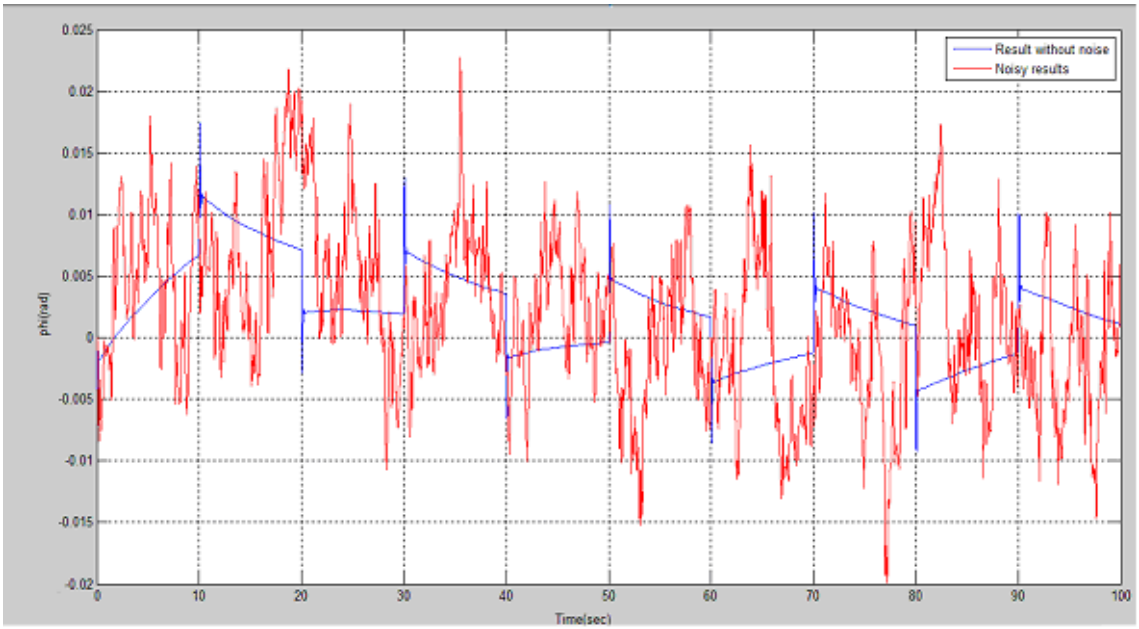



Figure 72: Response in φ for both cases

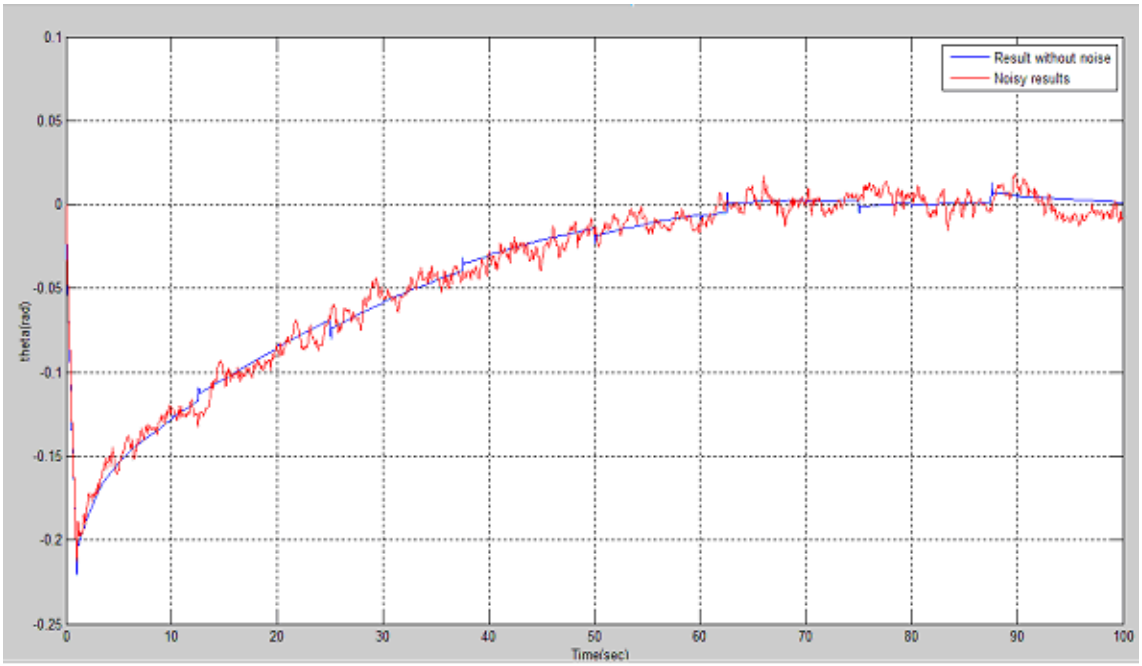



Figure 73: Response in θ for both cases

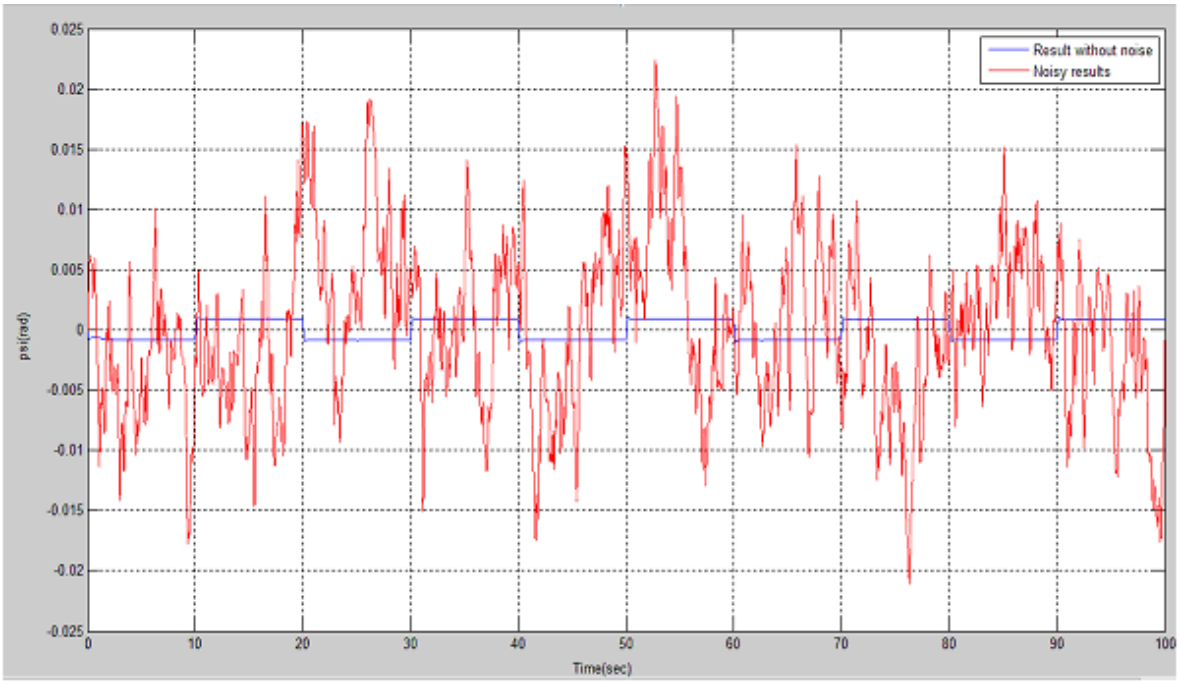



Figure 74: Response in ψ for both cases

It can be concluded that despite of the disturbances and measurement noise that are introduced in the system, the behaviour of it does not change, it continues being stable during all the simulation.

# ***Conclusion:***

After all this work, it can be said that Robust Control is a useful tool for the development of the representation of different kind of devices without the necessity of using the real one. Once that the correct simply representation of the model is got, different elements as wind disturbances and/or noise are possible to introduce in order to get a model which is more close to the reality, as can be seen in the last chapter.

As it is shown in first cases of the third chapter, the values of μ that are got does not ensure the robust stability and robust performance, but the responses that is get is faster than the case in which the value of μ is smaller than one. Furthermore, the values of the angles and position of the first cases are not so far of the trim values. So it is in the hands of the developer the decision of which of the controller is better for his/her work. If it is wanted to take worse response but faster the first one will be chosen and if it is wanted a slower responses but with more accuracy ensuring robust stability and robust performance the last one will be chosen.

It also should be noted that for the development of this work it is necessary to have some knowledge about the numerical stability, which appears in the third chapter in the design of the controller. There is possible to see how depending of the conditioning of the matrices, small changes made in the gains of the weighting functions got big changes in the values of the controller.

# ***Bibliography***


1. Petko H. Petkov, Da-Wei Gu, Mihail M. Konstantinov; Robust Control Design with *MATLAB*® .

2. L. Mollov , J. Kralev , T. Slavov , P. Petkov; μ-Synthesis and Hardware-in-the-loop Simulation of Miniature Helicopter Control System. J Intell Robot Syst manuscript No

3. Amr Nagaty, Sajad Saeedi, Carl Thibault , Mae Seto , Howard Li; Control and Navigation Framework for Quad-rotor Helicopters. J. Intell Robot System.

4. Gyou Beom Kim, Trung Kien Nguyen, Augs Budiyono, Jung Keun Park and Kwang Joon Yoon; Design and Development of a Class of Rotorcraft-based UAV. International Journal of Advanced Robotic Systems.

5. R. Czyba, G. Szafranski; Control Structure Impact on the Flying Performance of the Multi-Rotor VTOL Platform – Design, Analysis and Experimental Validation

6. P. Castillo, P. García, R. Lozano, P. Albertos: Modelado y Estabilización de un Helicóptero de Cuatro Rotores. Revista Iberoamericana de Automática e Informática Industrial. ISSN 1697-7912. Vol. 4 Num. 1.

# Annex

## Matlab code

### Frs_quad

```
% Frequency responses of the
rocket stabilization system
%
% closed-loop interconnection
sim_quad
clp = lft(sim_ic,K,4,9);
%
% closed-loop frequency response
ref_loop = clp(10:12,1:3);
omega = logspace(-2,2,200);
figure(1)
sigma(ref_loop,'b-',Wm,'r-
',omega), grid
title('Closed-loop singular
value plot')
%
% singular values of the output
sensitivity function
sen_loop = clp(10:12,4:6);
omega = logspace(-2,3,200);
figure(2)
sigma(sen_loop,'b-',inv(Wp),'r--
',omega), grid
title('Singular value plot of
the output sensitivity
function')
axis([10^(-2) 10^3 -400 100])
%
% singular values of the input
sensitivity function
cont_loop = clp([17:20],[4:6]);
omega = logspace(-2,3,200);
figure(3)
sigma(cont_loop,'b-',inv(Wu),'r-
-',omega), grid
title('Singular value plot of
the input sensitivity function')
axis([10^(-2) 10^3 -300 100])
%
% sensitivity of the control
effort to the reference
ref_u = clp(17:20,1:3);
omega = logspace(-2,3,200);
figure(4)
sigma(ref_u,'r-',omega), grid
title('Sensitivity of the
control to references')
%
% controller frequency responses
omega = logspace(-3,3,200);
figure(5)
sigma(K,'r-',omega), grid
title('Singular value plot of
the controller')
%
% open-loop frequency response
G_ang_u = sim_ic(10:12,7:10);
L = G_ang_u*K(1:4,4:9);
figure(6)
sigma(L.Nominal,'r-',L,'b--
',omega), grid
title('Singular value plot of
the open-loop system')
%legend('Nominal system','Random
samples',3)
```

### Mod_quad

```
% Uncertain quadcopter model
%
% Trim values
[x_trim,w_trim] = trim_val_quad
%[x_trim,w_trim] =
trim_val_quad2 % 13th order
model
G_quad =
num_lin_quad(x_trim,w_trim);
%G_quad =
num_lin_quad2(x_trim,w_trim); %
13th order model
%
% Model uncertainty
delt =
ultidyn('delt',1,'Bound',0.1); %
10% unceratinty
delta = 1 + delt;


Delta = [delta   0    0    0
0   0   0
```

```
            0   delta  0    0
0   0   0
            0     0  delta  0
0   0   0
            0     0    0  delta
0   0   0
            0     0    0    0
1   0   0
            0     0    0    0
0   1   0
            0     0    0    0
0   0   1];
```

```
G_quad_unc = G_quad*Delta; %
input multiplicative uncertainty
%
phi_trim     = x_trim(7);
theta_trim   = x_trim(8);
psi_trim     = x_trim(9);
%
U_ref1_trim  = w_trim(1);
U_ref2_trim  = w_trim(2);
U_ref3_trim  = w_trim(3);
U_ref4_trim  = w_trim(4);
%
U_max = 150;
```

### *Ms_quad*

```
% mu-synthesis of the quadcopter
stabilization system
%
nmeas = 9;
ncont = 4;
numit = 4;
%
fv = linspace(10^(-
3),2*10^2,100);
%fv = logspace(-3,2,100);
opt =
dkitopt('FrequencyVector',fv,
...
```

```
'DisplayWhileAutoIter','on', ...

'NumberOfAutoIterations',numit)
%opt =
dkitopt('DisplayWhileAutoIter','
on')
[K_mu,CL_mu,bnd_mu,dkinfo] =
dksyn(sys_ic,nmeas,ncont,opt);
K = K_mu;
[a_c,b_c,c_c,d_c] = ssdata(K);
```

### *Num_lin_quad*

```
function G_quad =
num_lin_quad(x_trim,w_trim)
%
% M-function to compute the
linearized quadcopter model
%
% Inputs:  x     - vector
components:
%                  x(1)      = u
Translatory velocities in BF
%                  x(2)      = v
%                  x(3)      = w
%                  x(4)      = p
Angular velocities in BF
%                  x(5)      = q
%                  x(6)      = r
%                  x(7)      =
phi       Euler angles
%                  x(8)      =
theta
%                  x(9)      =
psi
%
%          w     - input
signals:
```

```
%                  w(1)  =
U_ref1        Rotor control
torques
%                  w(2)  =
U_ref2
%                  w(3)  =
U_ref3
%                  w(4)  =
U_ref4
%                  w(5)  =
u_wind        Wind velocties
%                  w(6)  =
v_wind
%                  w(7)  =
w_wind
%
% Output: G_quad - state space
quadcopter model
%
x = x_trim;
u = w_trim(1:7);
%
% Linearization
[a,b,c,d] =
linmod('linmod_quad',x,u);
statenames  = {'u', 'v', 'w',
'p', 'q', 'r', ...
```

```
                '\phi', '\theta',
'\psi'};
inputnames  =
{'U_{ref1}','U_{ref2}','U_{ref3}
','U_{ref4}',...

'u_{wind}','v_{wind}','w_{wind}'
};
outputnames =
{'n_x','n_y','n_z','u','v','w',
...
'p','q','r','\phi','\theta','\ps
i',  ...
                '\Omega1',
'\Omega2', '\Omega3',
'\Omega4'};
G_quad =
ss(a,b,c,d,'StateName',statename
s,'InputName',inputnames, ...

'Outputname',outputnames);
```

***Olp_quad***

```
% Generates the open-loop
connection for the
% quadcopter control system and
displays
% frequency responses
%
% Quadcopter model
mod_quad
% Sensor models
wsa_quad
% Servo models
%servo_quad
% Weighting functions
wts_quad
%
systemnames = ' G_quad_unc Wm Wp
Wu ';
inputvar = '[ ref{3}; dist{3};
control{4} ]';
outputvar = '[ Wp; Wu; ref;
G_quad_unc(7:9);
G_quad_unc(10:12) ]';
input_to_G_quad_unc = '[
control; dist ]';
input_to_Wm = '[ ref ]';
input_to_Wp = '[
G_quad_unc(10:12) - Wm ]';
input_to_Wu = '[ control ]';
sys_ic = sysic;
%
% Quadcopter frequency responses
omega = logspace(-3,3,200);
figure(1)
%legend('Nominal plant','Random
samples',3)
sigma(G_quad(10:12,1:4),'r-
',G_quad_unc(10:12,1:4),'b--
',omega), grid on
title('Frequency responses in
respect to controls')
omega = logspace(-3,3,200);
%figure(2)
%legend('Nominal plant','Random
samples',3)
%sigma(G_heli(10:12,6:8),'r-
',G_heli_unc(10:12,6:8),'b--
',omega), grid on
%title('Frequency responses in
respect to disturbances')
%
omega = logspace(-1,5,200);
figure(3)
bode(wa,omega), grid
title('Accelerometer frequency
response')
%
omega = logspace(-1,5,200);
figure(4)
bode(wg,omega), grid
title('Rate gyro frequency
response')
%
omega = logspace(-2,3,200);
figure(5)
bodemag(wm1,'r-',wm2,'b--
',wm3,'m-.',omega), grid
title('Model frequency
response')
legend('wm1','wm2','wm3')
%
omega = logspace(-1,4,200);
figure(6)
bodemag(1/wp1,'r-',1/wp2,'b--
',1/wp3,'m-.',omega), grid
title('Inverses of Performance
Weighting Functions')
legend('1/wp1','1/wp2','1/wp3',2
)
%
omega = logspace(1,6,200);
figure(7)
%bodemag(1/wu1,'r-',omega), grid
bodemag(1/wu1,'r-',1/wu2,'b--
',1/wu3,'m-.',1/wu4,'c.',omega),
grid
title('Inverses of Control
Weighting Functions')
legend('1/wu1','1/wu2','1/wu3','
1/wu4',3)
```

***Pid_control***

```
function [sys,x0,str,ts] =
pid_control(t,x,w,flag)
%
% S-function for quadcopter PD
control
% P.Hr. Petkov, 01/04/2011
%
% Inputs:  t    - time in secs.
%
%          x    - state vector:
%                 x(1)  contains
x
%                 x(2)  contains
y
%                 x(3)  contains
z
%
%          w    - input signals:
%                 w(1)  contains
x_ref
%                 w(2)  contains
y_ref
%                 w(3)  contains
z_ref
%                 w(4)  contains
phi
%                 w(5)  contains
theta
%                 w(6)  contains
psi
%                 w(7)  contains
p
%                 w(8)  contains
q
%                 w(9)  contains
r
%                 w(10) contains
V_x
%                 w(11) contains
V_y
%                 w(12) contains
V_z
%
%          flag - an integer
value that indicates the task
%                 to be
performed by the S-function:
%                 flag = 0 -
initialize the state vector
%                 flag = 1 -
calculate the state derivatives
%                 flag = 3 -
calculate outputs
%
% Outputs: sys  - a generic
return argument whose values
depend
%                 on the flag
value.
%          x0   - the initial
state values. x0 is ignored,
except
%                 when flag = 0.
%          str  - argument
reserved for future use.
%          ts   - a two column
matrix containing the sample
times
%                 and offsets of
the blocks. For continuous time
%                 systems ts =
[0 0].
%
%
% Dispatch the flag
%
switch flag,

case 0
%
%   Initialization
%
%   Call function simsizes to
create the sizes structure.
    sizes = simsizes;
%   Load the sizes structure
with the initialization
information.
    sizes.NumContStates  = 3;
    sizes.NumDiscStates  = 0;
    sizes.NumOutputs     = 4;
    sizes.NumInputs      = 13;
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
%   Load the sys vector with the
sizes information.
    sys = simsizes(sizes);
%
%   Initialize the state vector
%
    x0 = [ 0
           0
          -5.];
%
%   str is an empty matrix.
%
    str = [];
%
    ts = [0 0];

case 1
%
%   Calculate derivatives
%
```

```
%   Inputs
    x_ref       = w(1);
    y_ref       = w(2);
    z_ref       = w(3);
    psi_ref     = w(4);
    phi         = w(5);
    theta       = w(6);
    psi         = w(7);
    p           = w(8);
    q           = w(9);
    r           = w(10);
    V_x         = w(11);
    V_y         = w(12);
    V_z         = w(13);
%
%   Position coordinates
    dxdt(1) = V_x;
    dxdt(2) = V_y;
    dxdt(3) = V_z;
%
    sys = [dxdt(1)  dxdt(2)
dxdt(3)]';

case 3
%
%   Calculate outputs
%
%   Inputs
    x_ref       = w(1);
    y_ref       = w(2);
    z_ref       = w(3);
    psi_ref     = w(4);
    phi         = w(5);
    theta       = w(6);
    psi         = w(7);
    p           = w(8);
    q           = w(9);
    r           = w(10);
    V_x         = w(11);
    V_y         = w(12);
    V_z         = w(13);
%
%   Position coordinates
    xx          = x(1);  %
    y           = x(2);
    z           = x(3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
%
%   Attitude and altitude
control
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
    z_ref_dot     = 0;
    phi_ref       = 0;
    theta_ref     = 0;
    phi_ref_dot   = 0;
    theta_ref_dot = 0;
    psi_ref_dot   = 0;
%
%   Altitude control law
    K_p1 = -1.0;
    K_d1 = -5.0;
    U1 = (K_p1*(z_ref - z) + ...

K_d1*(z_ref_dot -
V_z))/(cos(phi)*cos(theta));
%
%   Roll control law
    K_p2 = 2.5;
    K_d2 = 4.0;
    U2 = (K_p2*(phi_ref - phi) +
K_d2*(phi_ref_dot -
p))/cos(psi);
%
%   Pitch control law
    K_p3 = 2.5;
    K_d3 = 4.0;
    U3 = (K_p3*(theta_ref -
theta) + K_d3*(theta_ref_dot -
q))/ ...

(cos(phi)*cos(psi));
%
%   Yaw control law
    K_p4 = -8.0;
    K_d4 = -10.0;
    U4 = (K_p4*(psi_ref - psi) +
...

K_d4*(psi_ref_dot -
r))/(cos(phi)/cos(theta));
%
%   Calculate outputs
    U = [U1 U2 U3 U4]';
    V(1) = U(1) + U(3) + U(4);
    V(2) = U(1) - U(2) - U(4);
    V(3) = U(1) - U(3) + U(4);
    V(4) = U(1) + U(2) - U(4);
%
%   Converting radians to
degrees
    s = 180/pi;
    V = V*s;
%
    U_ref1 = V(1);
    U_ref2 = V(2);
    U_ref3 = V(3);
    U_ref4 = V(4);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
%
%   Position control
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
    x_ref_dot = 0;
    y_ref_dot = 0;
%
```

```
%   Y-control law
    K_p2 = 1.0;
    K_d2 = 5.0;
    U2 = (K_p2*(y_ref - y) +
K_d2*(y_ref_dot -
V_y))/cos(psi);
%
%   X-control law
    K_p3 = -1.0;
    K_d3 = -5.0;
    U3 = (K_p3*(x_ref - xx) +
K_d3*(x_ref_dot -
V_x))/(cos(phi)*cos(psi));
%
%   Calculate outputs
    U_ref1 = U_ref1 + U3;
    U_ref2 = U_ref2 - U2;
    U_ref3 = U_ref3 - U3;
    U_ref4 = U_ref4 + U2;
%
% Outputs:
    sys(1)  = U_ref1;   % Rotor
torques
    sys(2)  = U_ref2;
    sys(3)  = U_ref3;
    sys(4)  = U_ref4;

case { 2, 4, 9 }
%
%   Unused flags
%
    sys = [];
otherwise
    error(['Unhandled flag =
',num2str(flag)]); % Error
handling
end
% End of pid_control
```

### *Quad_model*

```
function [sys,x0,str,ts] =
quad_model(t,x,w,flag)
%
% S-function for nonlinear
simulation of a quadcopter
dynamics
% P.Hr. Petkov, 10/05/2011
%
% Inputs:  t    - time in secs.
%          x    - state vector:
%                 x(1)  contains
u
%                 x(2)  contains
v
%                 x(3)  contains
ww
%                 x(4)  contains
p
%                 x(5)  contains
q
%                 x(6)  contains
r
%                 x(7)  contains
phi
%                 x(8)  contains
theta
%                 x(9)  contains
psi
%          w    - input signals:
%                 w(1)  contains
U_ref1
%                 w(2)  contains
U_ref2
%                 w(3)  contains
U_ref3
%                 w(4)  contains
U_ref4
%                 w(5)  contains
u_wind
%                 w(6)  contains
v_wind
%                 w(7)  contains
w_wind
%          flag - an integer
value that indicates the task
%                 to be
performed by the S-function:
%                 flag = 0 -
initialize the state vector
%                 flag = 1 -
calculate the state derivatives
%                 flag = 3 -
calculate outputs
%
% Outputs: sys  - a generic
return argument whose values
depend
%                 on the flag
value.
%          x0   - the initial
state values. x0 is ignored,
except
%                 when flag = 0.
%          str  - argument
reserved for future use.
%          ts   - a two column
matrix containing the sample
times
%                 and offsets of
the blocks. For continuous time
%                 systems ts =
[0 0].
%
%
```

```
% Dispatch the flag
%
switch flag,

case 0
%
%   Initialization
%
%   Call function simsizes to
create the sizes structure.
    sizes = simsizes;
%   Load the sizes structure
with the initialization
information.
    sizes.NumContStates  = 9;
    sizes.NumDiscStates  = 0;
    sizes.NumOutputs     = 19;
    sizes.NumInputs      = 7;
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
%   Load the sys vector with the
sizes information.
    sys = simsizes(sizes);
%
%   Initialize the state vector
%
    x0 = inc_quad;
%
%   str is an empty matrix.
    str = [];
%
    ts = [0 0];

case 1
%
%   Calculate derivatives
%
%   Inputs
    U_ref1    = w(1);  % Motor
references
    U_ref2    = w(2);
    U_ref3    = w(3);
    U_ref4    = w(4);
    M_dx      = w(5);  % Wind
velocities in BF
    M_dy      = w(6);
    M_dz      = w(7);
%
%   State vector components
    u         = x(1);  %
Translatory velocities in BF
    v         = x(2);
    ww        = x(3);
    p         = x(4);  %
Angular velocities in BF
    q         = x(5);
    r         = x(6);
    phi       = x(7);  %  Euler
angles
    theta     = x(8);
    psi       = x(9);
%
    K_m = 5.389;
    Omega1 = K_m*U_ref1;
    Omega2 = K_m*U_ref2;
    Omega3 = K_m*U_ref3;
    Omega4 = K_m*U_ref4;
%
    b = 3.13*10^(-5);
    T1 = b*Omega1^2;
    T2 = b*Omega2^2;
    T3 = b*Omega3^2;
    T4 = b*Omega4^2;
    d_q = 7.5*10^(-7);
    Q1 = d_q*Omega1^2;
    Q2 = d_q*Omega2^2;
    Q3 = d_q*Omega3^2;
    Q4 = d_q*Omega4^2;
%
    g         = 9.80665;  %
m/s^2   acceleration of gravity
at sea level
    m         = 0.5;      % kg
quadcopter mass
    I_xx      = 0.01676;  %
kg.m^2
    I_yy      = 0.01676;  %
kg.m^2
    I_zz      = 0.02314;  %
kg.m^2
%
%   Forces in BF
    T = T1 + T2 + T3 + T4;
%%
%   Fuselage forces
    rho = 1.225;    % kg/m^3
    S_x_fus = 0.05; % m^2
    S_y_fus = 0.05; % m^2
    S_z_fus = 0.15; % m^2
    u_wind = 0;
    v_wind = 0;
    w_wind = 0;
    u_a = u - u_wind;
    v_a = v - v_wind;
    w_a = ww - w_wind;
    V_inf = sqrt(u_a^2 + v_a^2 +
w_a^2);
    X_fus = -
rho*S_x_fus*u_a*V_inf/2;
    Y_fus = -
rho*S_y_fus*v_a*V_inf/2;
    Z_fus = -
rho*S_z_fus*w_a*V_inf/2;
%
    X = X_fus;
    Y = Y_fus;
    Z = Z_fus - T;
%
%   Translatory velocities in BF
    dxdt(1) = v*r - ww*q -
g*sin(theta) + X/m;           %
du/dt
```

```
    dxdt(2) = ww*p - u*r +
g*sin(phi)*cos(theta) + Y/m;  %
dv/dt
    dxdt(3) = u*q - v*p  +
g*cos(phi)*cos(theta) + Z/m;  %
dw/dt
%
%   Translatory velocities in EF
    V_x = (cos(theta)*cos(psi)*u
+ (sin(phi)*sin(theta)*cos(psi)
- ...
           cos(phi)*sin(psi))*v
+ (cos(phi)*sin(theta)*cos(psi)
+ ...

sin(phi)*sin(psi))*ww);
    V_y = (cos(theta)*sin(psi)*u
+ (sin(phi)*sin(theta)*sin(psi)
+ ...
           cos(phi)*cos(psi))*v
+ (cos(phi)*sin(theta)*sin(psi)
- ...

sin(phi)*cos(psi))*ww);
    V_z = (-sin(theta)*u +
sin(phi)*cos(theta)*v + ...

cos(phi)*cos(theta)*ww);
%
%   Torques
    d       = 0.26;   % m
distance between the propeller
axis
                      %
and the c.g.
    I_rot   = 0.0001; % kg.m^2
rotor moment of inertia
    L       = -I_rot*q*(Omega1 -
Omega2 + Omega3 - Omega4) + d*(-
T2 + T4);
    M       =  I_rot*p*(Omega1 -
Omega2 + Omega3 - Omega4) +
d*(T1 - T3);
    N       = -Q1 + Q2 - Q3 +
Q4;
%
%   Angular velocities in BF
    dxdt(4) = q*r*(I_yy -
I_zz)/I_xx + L/I_xx + M_dx/I_xx;
% dp/dt
    dxdt(5) = p*r*(I_zz -
I_xx)/I_yy + M/I_yy + M_dy/I_yy;
% dq/dt
    dxdt(6) = p*q*(I_xx -
I_yy)/I_zz + N/I_zz + M_dz/I_zz;
% dr/dt
%
%   Euler angles
    dxdt(7) = p +
sin(phi)*tan(theta)*q +
cos(phi)*tan(theta)*r;% dphi/dt
    dxdt(8) = cos(phi)*q -
sin(phi)*r;
% dtheta/dt
    dxdt(9) =
(sin(phi)/cos(theta))*q +
(cos(phi)/cos(theta))*r;%
dpsi/dt
%
    sys = [dxdt(1)  dxdt(2)
dxdt(3)  dxdt(4)  dxdt(5)
dxdt(6) ...
          dxdt(7)  dxdt(8)
dxdt(9)]';
case 3
%
%   Calculate outputs
%
%   Inputs
    U_ref1    = w(1);  % Motor
references
    U_ref2    = w(2);
    U_ref3    = w(3);
    U_ref4    = w(4);
    M_dx      = w(5);  % Wind
velocities in BF
    M_dy      = w(6);
    M_dz      = w(7);
%
    u         = x(1);  %
Translatory velocities in BF
    v         = x(2);
    ww        = x(3);
    p         = x(4);  % Angular
velocities in BF
    q         = x(5);
    r         = x(6);
    phi       = x(7);  % Euler
angles
    theta     = x(8);
    psi       = x(9);
%
    K_m = 5.389;
    Omega1 = K_m*U_ref1;
    Omega2 = K_m*U_ref2;
    Omega3 = K_m*U_ref3;
    Omega4 = K_m*U_ref4;
%
    b = 3.13*10^(-5);
    T1 = b*Omega1^2;
    T2 = b*Omega2^2;
    T3 = b*Omega3^2;
    T4 = b*Omega4^2;
    d_q = 7.5*10^(-7);
    Q1 = d_q*Omega1^2;
    Q2 = d_q*Omega2^2;
    Q3 = d_q*Omega3^2;
    Q4 = d_q*Omega4^2;
%
    g    = 9.80665;      % m/s^2
acceleration of gravity at sea
level
```

```
    m    = 0.5;          % kg
    I_xx      = 0.01676;  %
kg.m^2
    I_yy      = 0.01676;  %
kg.m^2
    I_zz      = 0.02314;  %
kg.m^2
%
%   Forces in BF
    T = T1 + T2 + T3 + T4;
%%
%   Fuselage forces
    rho = 1.225;    % kg/m^3
    S_x_fus = 0.05; % m^2
    S_y_fus = 0.05; % m^2
    S_z_fus = 0.15; % m^2
    u_wind = 0;
    v_wind = 0;
    w_wind = 0;
    u_a = u - u_wind;
    v_a = v - v_wind;
    w_a = ww - w_wind;
    V_inf = sqrt(u_a^2 + v_a^2 +
w_a^2);
    X_fus = -
rho*S_x_fus*u_a*V_inf/2;
    Y_fus = -
rho*S_y_fus*v_a*V_inf/2;
    Z_fus = -
rho*S_z_fus*w_a*V_inf/2;
%
    X = X_fus;
    Y = Y_fus;
    Z = Z_fus - T;
%
%   Translatory accelerations in
BF
    dudt = v*r - ww*q -
g*sin(theta) + X/m;           %
du/dt
    dvdt = ww*p - u*r +
g*sin(phi)*cos(theta) + Y/m;  %
dv/dt
    dwdt = u*q - v*p  +
g*cos(phi)*cos(theta) + Z/m;  %
dw/dt
%
%   Translatory velocities in EF
    V_x = (cos(theta)*cos(psi)*u
+ (sin(phi)*sin(theta)*cos(psi)
- ...
           cos(phi)*sin(psi))*v
+ (cos(phi)*sin(theta)*cos(psi)
+ ...

sin(phi)*sin(psi))*ww);
    V_y = (cos(theta)*sin(psi)*u
+ (sin(phi)*sin(theta)*sin(psi)
+ ...
           cos(phi)*cos(psi))*v
+ (cos(phi)*sin(theta)*sin(psi)
- ...

sin(phi)*cos(psi))*ww);
    V_z = (-sin(theta)*u +
sin(phi)*cos(theta)*v + ...

cos(phi)*cos(theta)*ww);
%
% Outputs:
    sys(1)  = dudt;   %
Translatory accelerations in BF
    sys(2)  = dvdt;
    sys(3)  = dwdt;
    sys(4)  = u;      %
Translatory velocities in BF
    sys(5)  = v;
    sys(6)  = ww;
    sys(7)  = V_x;    %
Translatory velocities in EF
    sys(8)  = V_y;
    sys(9)  = V_z;
    sys(10) = p;      % Angular
velocities in BF
    sys(11) = q;
    sys(12) = r;
    sys(13) = phi;    % Euler
angles
    sys(14) = theta;
    sys(15) = psi;
    sys(16) = Omega1; % Rotor
speeds
    sys(17) = Omega2;
    sys(18) = Omega3;
    sys(19) = Omega4;

case { 2, 4, 9 }
%
%   Unused flags
%
    sys = [];
otherwise
    error(['Unhandled flag =
',num2str(flag)]); % Error
handling
end
% End of quad_model
```

***Sim_quad***

```
% Generates the open-loop
connection for the quadcopter
stabilization
% system simulation
%
% quadcopter model
mod_quad
% sensor models
wsa_quad
% servo models
%servo_quad
%
systemnames = ' G_quad_unc ';
inputvar = '[ ref{3}; dist{3};
control{4} ]';
outputvar = '[ G_quad_unc;
control; ref; G_quad_unc(7:9);
G_quad_unc(10:12) ]';
input_to_G_quad_unc = '[
control; dist ]';
sim_ic = sysic
```

***Trim_val_quad***

```
function [x_trim,w_trim] =
trim_val_quad
%
% Quadcopter trim values
%
%   State vector components
%   x(1)      = u
Translatory velocities in BF
%   x(2)      = v
%   x(3)      = w
%   x(4)      = p
Angular velocities in BF
%   x(5)      = q
%   x(6)      = r
%   x(7)      = phi       Euler
angles
%   x(8)      = theta
%   x(9)      = psi
%
x0 = [0.0 0.0 0.0 0 0 0 0 0 0]';
% hover  quad_model
%x0 = [0  0 0 0 0 0 0 0 0]'; %
hover
%x0 = [10 0 0 0 0 0 0 0 0]'; %
cruise
%
%   Control vector components
%   u(1)      = u_ref1
%   u(2)      = u_ref2
%   u(3)      = u_ref3
%   u(4)      = u_ref4
%
u0 = [85.0 85.0 85.0 85.0]';
ix = [1; 2; 3]';
iu = [1; 2; 3; 4]';
[x,u,y,dx] =
trim('trim_quad',x0,u0,[],ix,[],
[]);
%
u_wind_trim   = 0;
v_wind_trim   = 0;
w_wind_trim   = 0;
%
x_trim = x;
w_trim(1) = u(1);        % Rotor
control torques
w_trim(2) = u(2);
w_trim(3) = u(3);
w_trim(4) = u(4);
w_trim(5) = u_wind_trim; % Wind
velocities
w_trim(6) = v_wind_trim;
w_trim(7) = w_wind_trim;
```

***wsa_quad***

```
% Sensor models
%
% Accelerometer transfer
function
numWa = [1];
T_a = 4.55*10^(-4);   % f_a =
350 Hz
ksi_a = 0.707;
denWa = [T_a^2  2*T_a*ksi_a  1];
gainWa = 1.0;   % V/g
wa = gainWa*tf(numWa,denWa);
Wa = [wa 0  0
      0  wa 0
      0  0  wa];
%
% Rate gyro transfer function
numWg= [1];
T_g = 4.55*10^(-4);   % f_g =
350 Hz
ksi_g = 0.707;
denWg = [T_g^2  2*T_g*ksi_g  1];
gainWg = 1.0;
wg = gainWg*tf(numWg,denWg);
Wg = [wg 0  0
```

```
      0  wg 0
      0  0  wg]
```

***Wts_quad***

```
% Performance weighting function
for the quadcopter control
% system
%
% Model transfer function
nuWm1 = 1;
dnWm1 = [1.6^2  2.24   1];  % T
= 1.6, omega = 0.625, ksi = 0.7
%dnWm1 = [2.0^2  2.80   1];  % T
= 2.0, omega = 0.5,   ksi = 0.7
%dnWm1 = [2.5^2  3.50   1];  % T
= 2.5, omega = 0.5,   ksi = 0.7
dnWm1 = [3.0^2  4.20   1];  % T
= 3.0, omega = 0.5,   ksi = 0.7
gainWm1 = 1;
wm1 = gainWm1*tf(nuWm1,dnWm1);
%
nuWm2 = 1;
dnWm2 = [1.6^2  2.24   1];  % T
= 1.6, omega = 0.625, ksi = 0.7
%dnWm2 = [1.8^2  2.52   1];  % T
= 1.8, omega = 0.5,   ksi = 0.7
%dnWm2 = [2.0^2  2.80   1];  % T
= 2.0, omega = 0.5,   ksi = 0.7
%dnWm2 = [2.5^2  3.50   1];  % T
= 2.0, omega = 0.5,   ksi = 0.7
dnWm2 = [3.0^2  4.20   1];  % T
= 3.0, omega = 0.5,   ksi = 0.7
gainWm2 = 1;
wm2 = gainWm2*tf(nuWm2,dnWm2);
%
nuWm3 = 1;
dnWm3 = [1.8^2  2.52   1];  % T
= 1.8, omega = 0.556, ksi = 0.7
%dnWm3 = [2.0^2  2.80   1];  % T
= 2.0, omega = 0.5,   ksi = 0.7
%dnWm3 = [2.5^2  3.50   1];  % T
= 2.0, omega = 0.5,   ksi = 0.7
dnWm3 = [3.0^2  4.20   1];  % T
= 3.0, omega = 0.5,   ksi = 0.7
gainWm3 = 1;
wm3 = gainWm3*tf(nuWm3,dnWm3);
%
Wm = [wm1  0  0
      0  wm2  0
      0   0  wm3];
%
% Performance weighting function
nuWp1 = [10^(-3)  1      ];
dnWp1 = [10^(-3)  10^(-1)];
nuWp1 = [10^(-2)  1      ];
dnWp1 = [10^(-2)  10^(-1)];
gainWp1 = 1.3*10^(0);
wp1 = gainWp1*tf(nuWp1,dnWp1);
%
nuWp2 = [10^(-3)  1      ];
dnWp2 = [10^(-3)  10^(-1)];
nuWp2 = [10^(-2)  1      ];
dnWp2 = [10^(-2)  10^(-1)];
gainWp2 = 1.3*10^(0);
wp2 = gainWp2*tf(nuWp2,dnWp2);
%
nuWp3 = [10^(-3)  1      ];
dnWp3 = [10^(-3)  10^(-1)];
nuWp3 = [10^(-2)  1      ];
dnWp3 = [10^(-2)  10^(-1)];
gainWp3 = 1.3*10^(0);
wp3 = gainWp3*tf(nuWp3,dnWp3);
%
Wp = [wp1 0   0
      0  wp2  0
      0   0  wp3];
%
% Control action weighting
function
%
nuWu1 = [0.002     1];
dnWu1 = [0.000003  1];
gainWu1 = 0.00185*10^(0);
wu1 = gainWu1*tf(nuWu1,dnWu1);
%
nuWu2 = [0.002     1];
dnWu2 = [0.000003  1];
gainWu2 = 0.00185*10^(0);
wu2 = gainWu2*tf(nuWu2,dnWu2);
%
nuWu3 = [0.002     1];
dnWu3 = [0.000003  1];
gainWu3 = 0.00185*10^(0);
wu3 = gainWu3*tf(nuWu3,dnWu3);
%
nuWu4 = [0.002     1];
dnWu4 = [0.000003  1];
gainWu4 = 0.00185*10^(0);
wu4 = gainWu4*tf(nuWu4,dnWu4);
%
Wu = [wu1 0  0  0
      0  wu2 0  0
      0  0 wu3  0
      0  0   0 wu4];
```